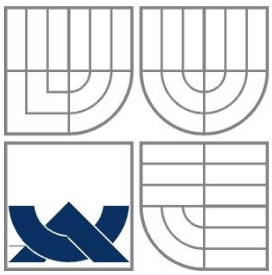
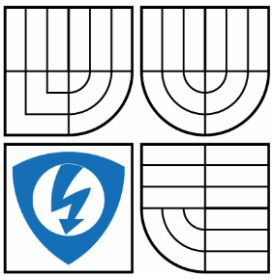


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

VYUŽITÍ TECHNIKY C2H PŘI IMPLEMENTACI ALGORITMŮ PRO FPGA

IMPLEMENTING ALGORITHMS ON FPGA UTILIZING C2H TECHNIQUE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LIBOR OTISK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. SOBĚSLAV VALACH

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student: Bc. Libor Otisk ***ID:*** 109705
Ročník: 2 ***Akademický rok:*** 2011/2012

NÁZEV TÉMATU:

Využití techniky C2H při implementaci algoritmů pro FPGA

POKYNY PRO VYPRACOVÁNÍ:

V rámci projektu vytvořte sadu algoritmů pro zpracování signálů ve struktuře hradlového pole. Pro vlastní implementace využijte nástroje typu C2H, která popisuje chování hardwaru v jazyce C.
V rozsahu potřebném pro řešení projektu je třeba nastudovat a pochopit následující body:
- Hradlové pole FPGA firmy Altera
- Logické obvody a sítě
- Jazyk VHDL a prostředí Quartu a SOPC Biulder
- Nástroj C2H compiler integrovaný v prostředí Quartus
- Algoritmy pro zpracování signálů

Dále ověřte funkci algoritmů na vývojovém kitu s hradlovým polem Altera. Především se jedná o implementaci algoritmů z oblasti zpracování jednorozměrných signálů (FIR, IIR). Porovnejte jednotlivé implementace z hlediska využití prostředků v FPGA, pracovní frekvence a vlastní složitosti implementace.

DOPORUČENÁ LITERATURA:

Kenneth L. Short, VHDL for Engineers
www.altera.com

Termín zadání: 6.2.2012 ***Termín odevzdání:*** 21.5.2012

Vedoucí práce: Ing. Soběslav Valach
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Tato práce se zabývá technikou C2H a jejím využití pří implementaci algoritmů pro FPGA. V rámci práce je implementováno s využitím C2H několik struktur číslicových filtrů FIR a IIR, u kterých je provedeno porovnání z hlediska využitých zdrojů FPGA, maximální frekvence, latence, složitosti implementace a získaného zrychlení vůči samotnému procesoru Nios II. Také je vytvořen příklad pro zpracování obrazu pomocí lokálních operátorů implementovaných s využitím C2H, umožňující zobrazení výsledného obrazu na LCD displeji.

Klíčová slova

C2H, FPGA, akcelerátor, FIR, IIR, lokální operátor

Abstract

This thesis deals with utilizing C2H technique for implementation algorithm on FPGA. Several structures of digital filters FIR and IIR are implemented within this work with usage of C2H. For such a comparison is in terms of FPGA resources utilized, the maximum frequency, latency, complexity of implementation and acceleration obtained to Nios II processor itself. Example for image processing using local operators implemented using C2h is also created to display the result on the LCD.

Keywords

C2H, FPGA, accelerator, FIR, IIR, local operator

Bibliografická citace:

OTISK, L. *Využití techniky C2H při implementaci algoritmů pro FPGA*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 76s. Vedoucí diplomové práce byl Ing. Soběslav Valach.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma Využití techniky C2H při implementaci algoritmů pro FPGA jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujícího autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **18. května 2012** ……

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Soběslavu Valachovi, za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **18. května 2012** ……

podpis autora

Obsah

1 Úvod 12
2 C-to-Hardvare Acceleratin (C2H) Kompilátor 13
2.1 Úvod 13
2.2 Vlastnosti C2H 13
2.3 Potřebné nástroje při použití techniky C2H 14
2.3.1 SOPC Builder 14
2.3.2 Quartus II 15
2.3.3 Nios II IDE 15
2.4 Rozhraní a komunikace hardwarového akcelerátoru 16
2.5 Přímý přístup do paměti 17
2.5.1 Propojovací direktiva 17
2.5.2 Restrict 18
2.6 Vhodný kandidát pro vytvoření akcelerátoru 18
3 Použitý vývojový kit 19
3.1 Jednotlivé části vývojového kitu 19
3.2 Hradlové pole Cyclone III EPC25F324 20
4 Zpracování obrazu 22
4.1 Lokální operátory 22
5 Lineární číslicové filtry 24
5.1 Úvod 24
5.2 Číslicové filtry s konečnou impulsní charakteristikou (FIR) 24
5.3 Základní struktury číslicových filtrů FIR 25
5.4 Filtry s nekonečnou impulsní charakteristikou (IIR) 26
5.5 Základní struktury číslicových filtrů IIR 27
5.5.1 Přímá struktura číslicového filtru IIR 27
5.5.2 Druhá kanonická struktura číslicového IIR filtru 28
5.5.3 První kanonická struktura číslicového filtru IIR 29
5.5.4 Kaskádní a paralelní struktura 30
6 Návrh číslicových filtrů pro implementaci v FPGA 33
6.1 Parametry navrhovaných číslicových filtrů FIR a IIR 33
6.2 Převod koeficientů filtrů do fixed-point tvaru 33
6.2.1 Reprezentace koeficientů filtrů ve fixed-point tvaru 34

6.2.2 Koeficienty u navrženého FIR filtru 35
6.2.3 Koeficientu u IIR filtrů 36
6.2.4 Koeficienty u kaskádní struktury IIR filtrů 38
7 Sestavené procesorové systémy 41
7.1 Použité komponenty 41
8 Implementace číslicových filtrů v FPGA pomocí techniky C2H 43
8.1 Realizace FIR filtru s jednou násobičkou 43
8.2 Paralelní realizace FIR filtru 46
8.3 Paralelní realizace FIR filtru s polovičním počtem násobiček 49
8.4 Realizace přímé struktury IIR filtru hardwarově neoptimalizované 52
8.5 Realizace přímé struktury IIR filtru optimalizované pro hardware 54
8.6 Realizace první kanonické struktury IIR filtru hardwarově neoptimalizovaného 56
8.7 Realizace první kanonické struktury IIR filtru optimalizované pro hardware 58
8.8 Realizace kaskádní struktury IIR filtru hardwarově neoptimalizovaného 59
8.9 Realizace kaskádní struktury IIR filtru optimalizovaného pro hardware 60
8.10 Realizace paralelní struktury IIR filtru 61
8.11 Testování funkčnosti implementovaných filtrů 63
8.12 Souhrn dosažených výsledků 63
9 Implementace hranového operátoru 65
10 Závěr 66
11 Použitá literatura 67

Seznam obrázků

Obrázek 2.1: Příklad připojení akcelerátoru k procesoru Nios II 16
Obrázek 3.1: Blokové schéma vývojového kitu NEEK převzato [4] 20
Obrázek 3.2: Hradlové pole Cyclone III EPC25F324 převzato a upraveno [5] 20
Obrázek 5.1: Přímá struktura FIR filtru 25
Obrázek 5.2: Struktura pro lineární fázovou charakteristiku pro sudé N 26
Obrázek 5.3: Struktura pro lineární fázovou charakteristiku pro liché N 26
Obrázek 5.4: Přímá struktura IIR filtru 28
Obrázek 5.5: Druhá kanonická struktura 29
Obrázek 5.6: První kanonická struktura 30
Obrázek 5.7: Kaskádní struktura filtru 31
Obrázek 5.8: Paralelní struktura filtru 31
Obrázek 6.1: Blokové schéma pro uložení velkého čísla s desetinou čárkou v FPGA 35
Obrázek 6.2: Srovnání frekvenčních charakteristiky FIR filtru 36
Obrázek 6.3: Srovnání frekvenčních charakteristik IIR filtru 37
Obrázek 6.4: Srovnání jednotkové kružnice IIR filtru 38
Obrázek 6.5: Srovnání frekvenčních charakteristik IIR filtru kaskádní struktury 39
Obrázek 6.6: Srovnání jednotkové kružnice IIR filtru kaskádní struktury 40
Obrázek 8.1: Akcelerátoru FIR filtru s jednou násobičkou 43
Obrázek 8.2: Paralelní realizace akcelerátoru FIR filtru 47
Obrázek 8.3: Paralelní realizace akcelerátoru FIR filtru s polovičním počten násobiček 50
Obrázek 8.4: Realizace přímé struktury IIR filtru bez hardwarové optimalizace 52
Obrázek 8.5: Realizace přímé struktury IIR filtru optimalizované pro hardware 55

Seznam tabulek

Tabulka 8.1: Specifické zdroje pro akcelerátor FIR filtru s jednou násobičkou 44
Tabulka 8.2: Využité zdroje pro FIR filtr s jednou násobičkou 44
Tabulka 8.3: Srovnání rychlosti pro FIR filtr s jednou násobičkou 45
Tabulka 8.4: Maximální frekvence pro FIR filtr s jednou násobičkou 45
Tabulka 8.5: Latence a CPLI pro FIR filtr s jednou násobičkou 45
Tabulka 8.6: Specifické hardwarové zdroje paralelní realizace akcelerátoru FIR filtru 47
Tabulka 8.7: Využité zdroje pro paralelní realizaci FIR filtru 48
Tabulka 8.8: Srovnání rychlosti pro paralelní realizaci FIR filtru 48
Tabulka 8.9: Maximální frekvence pro paralelní realizaci FIR filtru 48
Tabulka 8.10: Latence a CPLI pro paralelní realizaci FIR filtru..... 48
Tabulka 8.11: Specifické hardwarové zdroje FIR filtru s polovičním počtem násobiček 50
Tabulka 8.12: Využité zdroje pro FIR filtru s polovičním počtem násobiček 51
Tabulka 8.13: Srovnání rychlosti pro realizaci FIR filtru polovičním počtem násobiček 51
Tabulka 8.14: Maximální frekvence pro realizaci FIR filtru polovičním počtem násobiček 51
Tabulka 8.15: Latence a CPLI pro realizaci FIR filtru polovičním počtem násobiček..... 51
Tabulka 8.16: Specifické hardwarové zdroje přímé struktury IIR filtru bez optimalizace 53
Tabulka 8.17: Využité zdroje přímé struktury IIR filtru bez hardwarové optimalizace 53
Tabulka 8.18: Srovnání rychlosti přímé struktury IIR filtru bez hardwarové optimalizace 53
Tabulka 8.19: Maximální frekvence přímé struktury IIR filtru bez hardwarové optimalizace... 53
Tabulka 8.20: Latence a CPLI přímé struktury IIR filtru bez hardwarové optimalizace..... 54
Tabulka 8.21: Specifické hardwarové zdroje optimalizované přímé struktury IIR filtru 55
Tabulka 8.22: Využité zdroje optimalizované přímé struktury IIR filtru 55
Tabulka 8.23: Srovnání rychlosti optimalizované přímé struktury IIR filtru..... 56
Tabulka 8.24: Maximální frekvence optimalizované přímé struktury IIR filtru..... 56
Tabulka 8.25: Latence a CPLI optimalizované přímé struktury IIR filtru 56
Tabulka 8.26: Specifické hardwarové zdroje první kanonické struktury bez optimalizace..... 56
Tabulka 8.27: Využité zdroje první kanonické struktury IIR filtru bez optimalizace..... 57
Tabulka 8.28: Srovnání rychlosti první kanonické struktury IIR filtru bez optimalizace..... 57
Tabulka 8.29: Maximální frekvence první kanonické struktury IIR filtru bez optimalizace 57
Tabulka 8.30: Latence a CPLI první kanonické struktury IIR filtru bez optimalizace 57
Tabulka 8.31: Specifické hardwarové zdroje optimalizované první kanonické struktury 58
Tabulka 8.32: Využité zdroje optimalizované první kanonické struktury IIR filtru 58

Tabulka 8.33: Srovnání rychlosti optimalizované první kanonické struktury IIR filtru 58
Tabulka 8.34: Maximální frekvence optimalizované první kanonické struktury IIR filtru 59
Tabulka 8.35: Latence a CPLI optimalizované první kanonické struktury IIR filtru 59
Tabulka 8.36: Specifické hardwarové zdroje kaskádní struktury bez optimalizace 59
Tabulka 8.37: Využité zdroje kaskádní struktury IIR filtru bez optimalizace 59
Tabulka 8.38: Srovnání rychlosti kaskádní struktury IIR filtru bez optimalizace 60
Tabulka 8.39: Maximální frekvence kaskádní struktury IIR filtru bez optimalizace..... 60
Tabulka 8.40: Latence a CPLI kaskádní struktury IIR filtru bez optimalizace..... 60
Tabulka 8.41: Specifické hardwarové zdroje optimalizované kaskádní struktury..... 60
Tabulka 8.42: Využité zdroje optimalizované kaskádní struktury IIR filtru 61
Tabulka 8.43: Srovnání rychlosti optimalizované kaskádní struktury IIR filtru..... 61
Tabulka 8.44: Maximální frekvence optimalizované kaskádní struktury IIR filtru..... 61
Tabulka 8.45: Latence a CPLI optimalizované kaskádní struktury IIR filtru 61
Tabulka 8.46: Specifické hardwarové zdroje paralelní struktury..... 62
Tabulka 8.47: Využité zdroje paralelní struktury IIR filtru 62
Tabulka 8.48: Srovnání rychlosti paralelní struktury IIR filtru..... 62
Tabulka 8.49: Maximální frekvence paralelní struktury IIR filtru..... 62
Tabulka 8.50: Latence a CPLI paralelní struktury IIR filtru..... 63

1 ÚVOD

Neustálým vývojem programovatelných logických obvodů dochází k zvyšování jejich kapacity a složitosti. Současně je také snaha o vývoj nových návrhových nástrojů, které umožňují popis hardwaru na vyšší úrovni abstrakce. Tyto nástroje mají za úkol zvýšit produktivitu návrhářů, kteří se již nemusí starat o detaily na nejnižší úrovni návrhu a usnadňují jim vytváření rozsáhlých návrhů. Jednu z možností popisu hardwaru na vyšší úrovni přinášejí nástroje typu C-to-gates, které využívají k popisu samostatných hardwarových modulů programovací jazyk C. Odlišný přístup pro popis hardwaru na vyšší úrovni abstrakce pomocí jazyka C je vytváření hardwarových akcelerátorů připojených k soft-core procesoru. Tento přístup využívá technika C2H vyvinutá firmou Altera.

Touto technikou a jejím využitím pro implementaci základních algoritmů pro zpracování signálů se zabývá tato práce. V rámci, které má být implementováno s využitím techniky C2H několik struktur číslicových filtrů FIR a IIR. Takto implementované struktury mají být porovnány z hlediska využitých zdrojů FPGA, maximální frekvence, latence, složitosti implementace a získaného zrychlení vůči samotnému procesoru Nios II.

První kapitola práce popisuje vlastní C2H, jeho vlastnosti a nástroje potřebné při jeho využití. V další kapitole je popsán použitý vývojový kit spolu s FPGA, které obsahuje. Následující tři kapitoly se zabývají zpracováním obrazu pomocí lokálních operátorů, lineárními číslicovými filtry a jejich úpravami pro implementaci v FPGA. Další kapitola pak popisuje komponenty sestavených procesorových systémů. Poslední dvě kapitoly se zabývají vlastní implementací algoritmů.

2 C-TO-HARDVARE ACCELERATIN (C2H) KOMPILÁTOR

2.1 Úvod

S vývojem programovatelných logických obvodů došlo k značnému rozšíření používání soft-core procesorů. Tyto procesory jsou vytvářeny v programovatelném hradlovém poli (FPGA) ze základních prvků. Současně s vývojem soft-core procesorů vznikly i nové návrhové nástroje, umožňující snadné použití a připojení soft-core procesoru k ostatním modulům vytvořených v FPGA. Jedná se například o SDRAM řadič, vstupní/výstupní rozhraní a další části se kterými má procesor komunikovat. Použití soft-core procesoru přináší oproti klasickým mikroprocesorům nové možnosti jak dosáhnout zvýšení výkonu. Jedná se především o rozšíření procesoru o vlastní hardwarovou část, která zpracovává náročné operace v hardwaru mimo soft-core, případně vytvoření více spolupracujících procesorových jader v FPGA.

Pro snadné rozšíření soft-core procesoru o hardwarovou část vytvořila firma Altera nástroj C-to-Hardvare-Acceletarion (C2H) kompilátor. Tento nástroj umožňuje vytváření hardwarových akcelerátorů z funkcí napsaných v jazyce ANSI C. Hardwarový akcelerátor je typ koprocesoru, který pomáhá hlavnímu procesoru s výpočetně náročnými úkoly pro procesor, ale dobře řešitelnými v hardwaru. Cílem C2H tedy není vytváření libovolného hardwarového modulu z kódu napsaného v jazyce ANSI C, který funguje samostatně. Ale vytvoření hardwarových akcelerátorů, které jsou vždy připojeny a řízeny procesorem Nios II.

Pro vytváření samostatných hardwarových modulů z kódu napsaného v jazyce C, který je v omezené nebo modifikované formě, existuje značné množství nástrojů typu C-to-gates. Jako je například Forte Cynthesizer, Impulse-C, AutoESL‘s AutoPilot a další.

2.2 Vlastnosti C2H

U nástrojů typu C-to-gates je jedním z největších nedostatků absence možnosti použití ukazatelů pro přístup do paměti. S tím často souvisí i modifikace použitého jazyku C. Oproti tomu C2H kompilátor umožňuje akcelerátoru přímý přístup k pamětím, ke kterým může přistupovat procesor Nios II. Současně je také využíváno standartní ANSI C, není tedy vyžadovaná speciální syntaxe nebo knihovní funkce pro specifikování hardwarové struktury. I když je pro popis hardwaru použit programovací jazyk ANSI C, je nutné držet se při psaní kódu určitých pravidel, které jsou specifická pro popis hardwaru. Nicméně je možné pro vytvoření akcelerátoru použít kód, který je strukturou určen pro procesor. Tím lze dosáhnout určitého zrychlení u mnoha typů algoritmů. Ale pro maximální využití této techniky a dosažení maximálního zrychlení je

nutné vědět jak C2H kompilátor převádí C kód do hardwaru. A podle požadavků na strukturu hardwarového akcelerátoru přizpůsobit i strukturu kódu.[1]

Tímto postupem je umožněno ovlivňovat latenci a určovat části, které mají být vykonávány paralelně. C2H však neumožňuje definovat logiku s požadavky na složitější časování. Například není možné vytvořit libovolný stavový automat, který garantuje specifické operace na určitých hodinových cyklech. Nejedná se tedy o náhradu návrhu pomocí HDL jazyků.[1]

2.3 Potřebné nástroje při použití techniky C2H

V podkapitole jsou popsány nástroje SOPC Builder, Quartus II a Nios II IDE.

2.3.1 SOPC Builder

Vlastní C2H kompilátor je integrován ve vývojovém prostředí Nios II IDE. Jedná se o nástroj pro tvorbu softwaru pro procesor Nios II. Před vlastním spuštěním Nios II IDE je potřeba sestavit procesorový systém. Ten se sestavuje v SOPC Builderu (Systém on a Programmable Chip Builder) případně v Qsys, který je novou generací SOPC Builderu. Tyto nástroje jsou součástí softwaru Quartus II, ten je určen pro analýzu a syntézu HDL projektů. Sestavení systému v SOPC Builderu se provádí z komponent obsažených v knihovně, jedná se například o vlastní soft-core procesor Nios II, paměťové řadiče, rozhraní a periferie, komponenty pro obrazové zpracování a další. U většiny těchto komponent je možné provádět nastavení, kterými lze komponentu přizpůsobit konkrétnímu vytvářenému systému. SOPC Builder také umožňuje přidávání vlastních komponent do knihovny. Vybrané komponenty pro daný systém se pak v grafickém prostředí propojí podle požadavku na daný systém a přiřadí se jim adresa.

SOPC Builder z takto sestaveného systému vygeneruje HDL soubory, které popisují všechny použité komponenty a top-level HDL soubor, který je spojuje dohromady. Také je generován příklad instance nejvyšší úrovně, ke které se již přiřazují jednotlivé piny obvodu. Tyto soubory mohou být vygenerované ve Verilogu nebo VHDL. Komponenty, které však nejsou přímo v knihovně a přidávají se jako open-source podporují většinou pouze Verilog. Také je generován soubor pro časovou analýzu (.sdc), Quartus II IP (.qip) obsahující informace o generovaných HDL souborech a souborech pro časovou analýzu, Block Simbol File (.bsf) reprezentující top-level vytvořeného systému pro případné použití schématického editoru v Qurtus II. Pokud to použité komponenty umožňují, tak je možné nechat vygenerovat i test bench pro simulaci vytvořeného systému v softwaru ModelSim. Další generovaný soubor je SOPC information file (.sopcinfo), ten poskytuje úplný popis systému, jedná se o soubor, který je využívaný vývojovým prostředím Nios II IDE. Kromě těch to souborů je při použití procesoru Nios II vygenerován hlavičkový soubor obsahující adresy

jednotlivých komponent, případně jednotlivé komponenty mohou poskytovat softwarové ovladače, tyto soubory jsou pak využity při psaní programu.[2]

2.3.2 Quartus II

Jedná se o úplný návrhový nástroj, který obsahuje veškeré potřebné nástroje pro vytvoření návrhu pro FPGA. Quartus II umožňuje přistupovat k návrhu několika způsoby. Jde především o popis v HDL jazyce případně použití již zmíněného SOPC Builderu nebo schématického popisu. Jakmile je návrh kompletní, provede se syntéza a plánování pro rozmístění zdrojů a cest pro zvolené FPGA. Následně je možné využít integrovaných nástrojů pro analýzu. Jedná se například o časovou analýzu pro zjištění časově kritických cest, případně o power analýzu. V našem případě, kdy systém vytváříme v SOPC Builderu stačí v softwaru Quartus II pouze vytvořit projekt pro konkrétní FPGA. Pro takový projekt se vytvoří, případně vybere z generovaných souborů SOPC Buildrem top-level, ke kterému se přiřadí jednotlivé piny hradlového pole. Po zkompilování takto sestaveného projektu je možné provést vlastní konfiguraci hradlového pole.

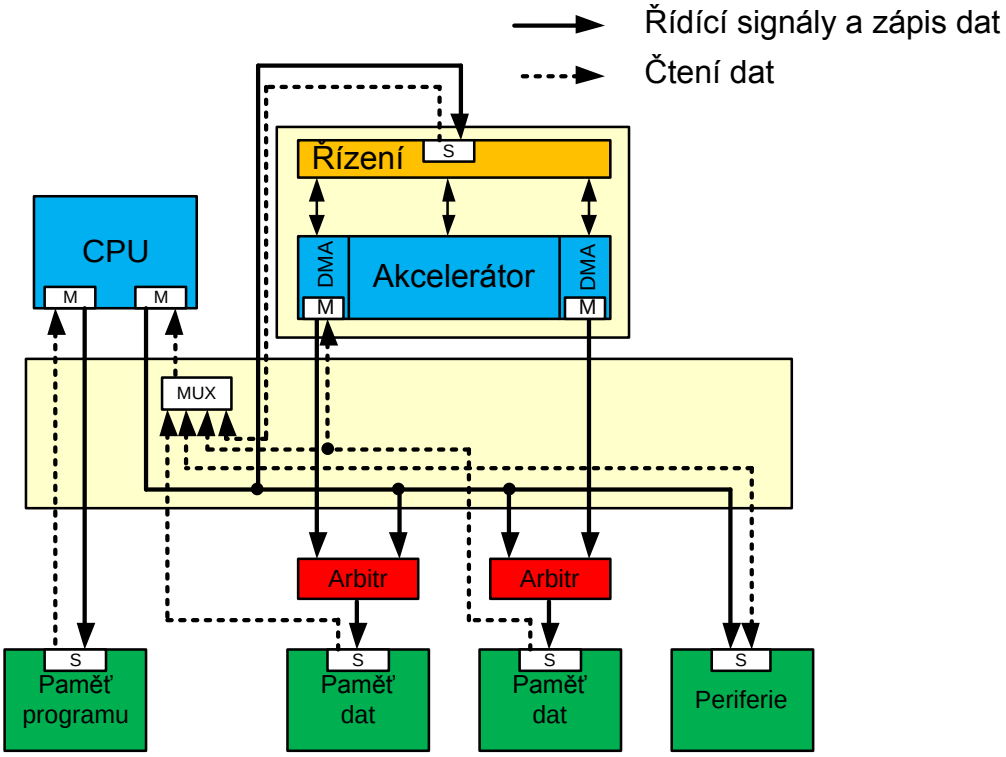
2.3.3 Nios II IDE

Jak již bylo zmíněno vlastní C2H kompilátor je integrovaný ve vývojovém softwaru Nios II IDE, který slouží k vytváření programů pro soft-core procesor Nios II. Integrace C2H do vývojového prostředí Nios II IDE umožňuje rychlé přejití mezi funkcí, která běží na soft-core procesoru na hardwarový akcelerátor případně obráceně. Tím je umožněno provést odladění kódu nejprve v softwaru. Po odladění je možné funkci nechat vytvořit jako akcelerátor. To se provede pouhým přepnutí pro použití Nios II C2H kompilátoru u dané funkce.

Po spuštění kompilace, C2H kompilátor provede u zvolené funkce předzpracování kódu a jeho analýzu. Následuje vytvoření grafu závislostí mezi jednotlivými částmi kódu a jejich optimalizace. Dále je určeno nejlepší pořadí, ve kterém se mají jednotlivé operace vykonávat a jsou vygenerovány soubory popisující akcelerátor. Také jsou vygenerovány C wrapper funkce, které obsahují detaily, o tom jak procesor komunikuje s konkrétním hardwarovým akcelerátorem. Tyto wrapper funkce nahradí původní funkce v programu, který běží na procesoru. Vlastní připojení hardwarového akcelerátoru k procesoru a ostatním periferiím je prováděno automaticky pomocí SOPC Builderu. Nios II IDE pak umožňuje provést konfiguraci hradlového pole a nahrát vytvořený program pro procesor Nios II do zvolené paměti.[1]

2.4 Rozhraní a komunikace hardwarového akcelerátoru

Každý hardwarový akcelerátor má slave port, který slouží jako řídící rozhraní pro procesor. Tento port obsahuje start bit, status bit a banku registrů pro přijímání vstupních argumentů akcelerované funkce. Pokud funkce vrací návratovou hodnotu, obsahuje také read-only registr. Procesor přistupuje k těmto registrům pouhým vydáním operace čtení nebo zápisu na dané místo. Na obrázku 2.1 je znázorněn jednoduchý příklad připojení hardwarového akcelerátoru k procesoru, pamětem dat a periferním zařízením. Z důvodu přehlednosti jsou na obrázku rozlišovány pouze dva typy vodičů a to pro čtení dat a druhý zahrnuje vodiče pro zápis dat a řídící signály.[3]



Obrázek 2.1: Příklad připojení akcelerátoru k procesoru Nios II

Vlastní komunikace mezi procesorem a hardwarovým akcelerátorem může pracovat ve dvou odlišných režimech, v prvním režimu se jedná o dotazovací řízení, ve druhém je využíváno přerušení. V dotazovacím režimu pracuje akcelerátor podle následujících kroků. Nejprve se načtou vstupní argumenty do registrové banky akcelerátoru, následuje nastavení start bitu, v dalším kroku je prováděno dotazování na status bit, po zpracování je návratová hodnota přečtena procesorem. Po opuštění akcelerované funkce, procesor již běžným způsobem pokračuje v programu. V režimu řízeným přerušením je načtení vstupních argumentů do registru banky akcelerátoru a nastavení stop bitu stejné jako v dotazovacím režimu, zbývající části jsou však odlišné. Překladač

rozšíří port akcelerátoru o hardwarové přerušení, současně je vygenerován hlavičkový soubor obsahující makra pro mazání přerušení, kontrolu stavu a čtení návratové hodnoty funkce. Použitím těchto maker mohou být napsány obslužné rutiny přerušení, které umožní souběžný běh procesoru s více akcelerátory.[3]

2.5 Přímý přístup do paměti

Jak již bylo zmíněno, přístup do paměti pomocí ukazatelů nebo polí je u nástrojů typu C-to-gates obtížně realizovatelné. Tato kapitola popisuje, jak je tento problém řešen u techniky C2H. Pro převod ze softwaru do hardwarového akcelerátoru případně obráceně musí být umožněno přistupovat do stejné paměti jako při softwarové realizaci, stejně tak i k ostatním komponentám systému. Pokud akcelerovaná funkce obsahuje ukazatel, pole nebo globální proměnou, které přistupují do paměti, C2H kompilátor vygeneruje Avalon master port. Tento master port umožňuje přístup do pamětí a k ostatním komponentám procesorového systému. Jednotlivé Avalon master porty jsou na sobě nezávislé a mohou tedy pracovat paralelně. V případě, kdy jednotlivé Avalon master porty přistupují k stejnému slave portu, je přístup prováděn sekvenčně a je vygenerován pouze jeden Avalon master port. Pokud je vyloučeno paralelní použití master portu, tak může dojít k jejich sloučení.[3]

2.5.1 Propojovací direktiva

Propojovací direktiva zajišťuje spojení Avalon master portu s Avalon slave portem, což je obvykle paměť. Pokud není připojovací direktiva použita C2H překladač defaultně propojí všechny master porty akcelerátoru ke všem paměťovým slave portům, ke kterým je připojen master port CPU. Takovéto defaultní nastavení připojení garantuje, že akcelerátor může přistupovat k jakékoliv paměťové adrese, ke které může přistupovat procesor. Nevýhodou defaultního nastavení připojení je, že s každým master-slave propojením souvisí multiplexování navíc a nárůst využité logiky na arbitráž. S tím často také souvisí snížení maximální frekvence. Z těchto důvodů je vhodné překladači prostřednictvím direktivy říct k jaké paměti má být daný master port připojen. Pro spojení master portu s více slave porty lze využít vícenásobnou direktivu.[1]

Direktiva, která spojuje akcelerátor s pamětí SSRAM přes tri_state_bridge je uvedena pod odstavcem. Přitom fir_hw je název akcelerované funkce, input_data je název ukazatele v uvedené funkci, který přistupuje do paměti a je připojen k tri_state_bridge přes který přistupuje do pamětí SSRAM.

```
#pragma altera_accelerate connect_variable fir_hw/input_data to
tri_state_bridge

#pragma altera_accelerate connect_variable fir_hw/output_data to
tri_state_bridge
```

2.5.2 Restrict

Při použití více ukazatelů v kódu pro hardwarový akcelerátor, C2H překladač připouští, že ukazatelé mohou ukazovat na stejné místo. V případě že pak mají být prováděny dvě operace paralelně, které využívají ukazatelů, je nejprve provedena ta operace, která se nachází v kódu jako první až po jejím dokončení je provedena druhá operace. Řešením jak provést obě operace paralelně je použití klíčového slova `__restrict__`. To garantuje, že nikdy nedojde k přepsání místa v paměti, na které ukazuje takto označený ukazatel jiným ukazatelem.[1]

```
short * __restrict__ input_data;
short * __restrict__ output_data;
```

2.6 Vhodný kandidát pro vytvoření akcelerátoru

Kód vhodný pro realizaci jako hardwarový akcelerátor je zpravidla takový, jehož vykonáváním stráví procesor značné množství času. Nejlepší kandidáti jsou jednoduché smyčky nebo několik vnořených smyček., které provádí každou iteraci určité operace. Výhodou je pokud lze operace uvnitř smyčky paralyzovat a tím dosáhnout vyššího zrychlení. Vhodné algoritmy pro použití hardwarového akcelerátoru jsou například z oblasti šifrování/dešifrování, filtrace, výpočtů, zpracování obrazu a dalších. V této práci budou především implementovány algoritmy pro filtraci 1D signálů a zpracování obrazu.[1]

Pokud je použit nevhodný kód pro vytvoření akcelerátoru může dojít k příliš velké spotřebě zdrojů FPGA nebo v krajním případě dokonce k snížení výkonu. Nevhodné kódy pro vytvoření akcelerátoru jsou takové, které obsahují značné množství závislostí. To způsobuje sekvenční vykonávání operací vhodných spíše pro procesor. Pro hardwarovou realizaci také nejsou vhodné kódy, v kterých se často přistupuje ke globálním nebo externím proměnným. V některých případech se může stát, že z důvodu nepodporované syntaxe akcelerátor nepůjde vytvořit. To může nastat, například pokud je v akcelerované funkci použit datový typ s plovoucí řádovou čárkou nebo se jedná o rekurzivní funkci.[1]

3 POUŽITÝ VÝVOJOVÝ KIT

Pro vlastní otestování funkčnosti jednotlivých algoritmů byl použit vývojoví kit Nios II Embedded Evaluation Kit (NEEK) od firmy Altera. Jedná se o vývojový kit Cyclone III FPGA Starter Kit rozšíření o multimediální kartu, která je připojena pomocí HSMC konektoru. Vlastní vývojový kit je osazen hradlovým polem Cyclone III. Na obrázku 3.1 je zobrazeno zjednodušené blokové schéma vývojového kitu.

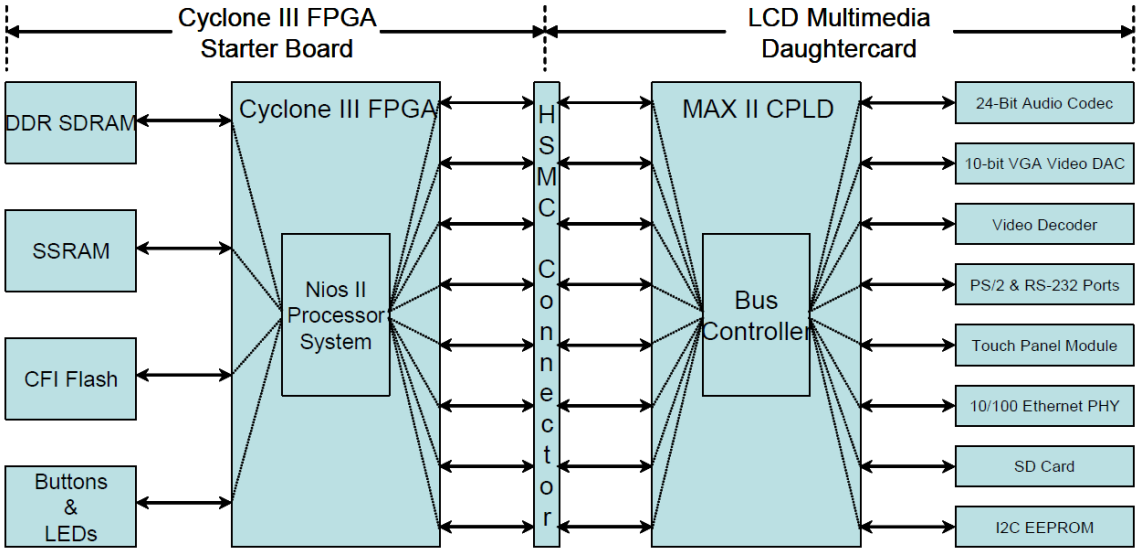
3.1 Jednotlivé části vývojového kitu

Vývojový kit Cyclone III FPGA Starter se skládá z následujících hlavních částí:

- Cyclone III EPC25F324 FPGA
- Konfigurační port USB
- 32 MB DDR SDRAM paměť
- 1 MB synchroní SRAM paměť
- 16 MB Intel P30/P33 flash paměť
- Tlačítka a LED

Připojením multimediální karty pak dojde k rozšířením o následující částí:

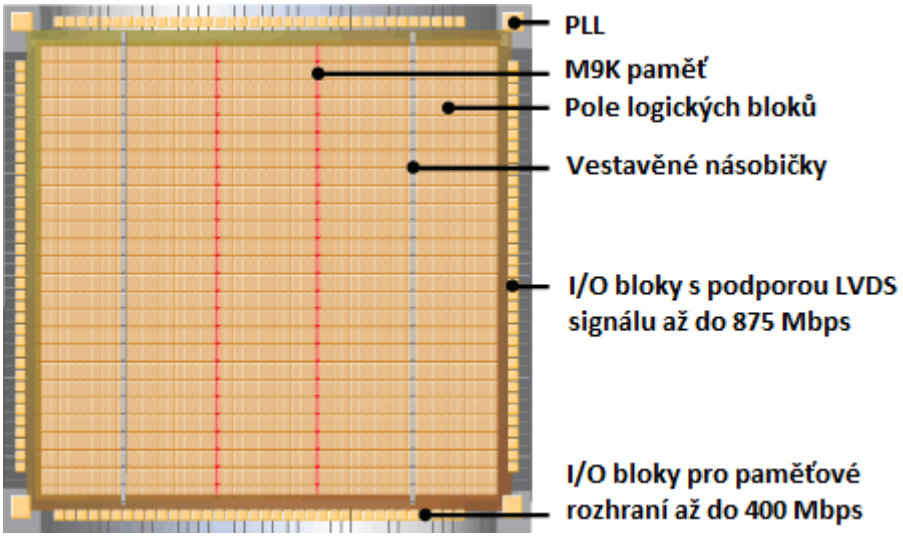
- LCD dotykový display s rozlišením 800 x 480
- 24-bitový audio CODEC
- Video dekodér
- 10-bitový VGA převodník
- 10/100 Ethernet rozhraní
- Konektory
    - VGA výstup
    - Kompozitní TV vstup
    - Audio line-in, line-out a microphone-in porty
    - Slot pro SD kartu
    - RS-232 sériový port
    - PS/2 port
    - Ethernet RJ-45 port



Obrázek 3.1: Blokové schéma vývojového kitu NEEK převzato [4]

3.2 Hradlové pole Cyclone III EPC25F324

Programovatelné hradlové pole jsou obvody, složené z mnoha logických elementů pomocí, kterých lze realizovat jednoduché logické funkce. Správným propojení takovýchto buněk lze docílit realizace složitých funkcí. K propojování jednotlivých bloků v hradlovém poli jsou určeny propojovací vodiče, které jsou pomocí programovatelných přepínačů příslušně propojovány. Hradlová pole dále často obsahují specifické bloky, u Cyclene III se především jedná o paměťové bloky, vestavěné násobičky, PLL . Umístění jednotlivých bloků obsažených v hradlovém poli Cyclone III je naznačeno na obrázku 3.2.



Obrázek 3.2: Hradlové pole Cyclone III EPC25F324 převzato a upraveno [5]

Logický element a logický blok

Základním prvkem hradlového pole je logický blok (LAB), který je v případě hradlového pole Cyclone III složen z šestnácti logických elementů (LE) a řídícího bloku. Logický element je nejmenší jednotkou logiky v hradlovém poli, je složen z čtyřvstupové vyhledávací tabulky (lookup table - LUT), ke které je připojen jeden klopný obvod. Vlastní logický blok umožňuje realizaci jednoduchých sekvenčních a kombinačních obvodů případně může fungovat jen jako propojovací člen.[6]

I/O bloky

Speciálním logickým blokem je vstupně výstupní blok (I/O blok). Tyto bloky obklopují ostatní logické bloky, které jsou uspořádány ve dvojrozměrné matici, tak jak je zobrazeno na obrázku 3.2. Jejich úkolem je zajistit spojení mezi vnitřní částí hradlového pole s jeho okolím.

Paměťový blok

Paměťové bloky obsažené v hradlovém poli jsou typu M9K. Jednotlivé bloky jsou uspořádány do dvou sloupců a mohou být konfigurovány jako RAM, FIFO nebo ROM paměť. V použitém hradlovém poli je těchto paměťových bloků 66. Každý paměťový blok umožňuje uložit devět kbitů. Paměťové moduly M9K, podporují pracovní režimy single-port, jednoduchý dual-port, dual-port a jsou schopny procovat na frekvenci 315MHz.[6]

Vestavěné násobičky

Vestavěné násobičky se používají v hradlových polích z důvodu požadavku na rychlé provedení násobící operace a snížení spotřeby zdrojů, které by se spotřebovaly na jejich vytvoření. V použitém hradlovém poli je 66 násobících bloků, jeden násobící blok přitom podporuje jednu 18x18 bitovou násobičku nebo dvě 9x9 bitové násobičky.[6]

PLL a cesty pro hodiny

V hradlovém poli Cyclone III jsou obsaženy čtyři bloky PLL. Tyto bloky umožňují vstupní hodinový signál násobit, dělit nebo fázově posunout. Každý PLL blok obsahuje pět výstupů. Pro rozvod vlastního hodinového signálu je v hradlovém poli obsaženo dvacet globálních sítí.[6]

4 ZPRACOVÁNÍ OBRAZU

Metody pro zpracování obrazu lze rozdělit podle použitého operátoru na globální, lokální a bodové. Při použití globálního operátoru je výstupní pixel upraveného obrazu ovlivněn všemi pixely vstupního obrazu. U lokálního operátoru je výstupní pixel ovlivněn pouze omezeným počtem pixelů vstupního obrazu. Výstupní pixel bodového operátoru je ovlivněn pouze pixelem na stejné pozici vstupního obrazu. Bodové operátory slouží především pro transformace kontrastu, úpravě zastoupených barev v obraze a dalším jednoduchým úpravám. [7]

Následující podkapitola se již zabývá pouze lokálními operátory, využitými v této práci.

4.1 Lokální operátory

Lokální operátory využívají pro vlastní zpravování obrazu masku nejčasněji o rozměrech 3x3, velikost masky ale může být i jiná. Tato maska je postupně posouvána po originálním obraze vždy o jeden pixel. V případě použití lineárních lokálních operátorů je hodnota výstupního pixelu daná součtem hodnot získaných vynásobením prvků masky s pixely originálního obrazu, nad kterými se maska v daný okamžik nachází. Matematicky se jedná o 2D diskrétní konvoluci, která je daná vztahem (1). V tomto vztahu je výstupní pixel značen jako *g*, váhová maska posouvaná po originálním obraze *f* je značena jako *h*.[7]

$$g(i,k) = \sum_{m=i-\frac{M}{2}}^{i+\frac{M}{2}} \sum_{n=k-\frac{N}{2}}^{k+\frac{N}{2}} f(m,n) \cdot h(m-i, n-k) \qquad (1)$$

Nejčastější využití lokálních operátorů je pro potlačení šumu v obraze, zostření obrazu případně pro detekci hran.

Operátory pro potlačení šumu

Pro potlačení šumu, který působí na všechny části obrazu stejně, se využívají průměrující operátory. Tyto operátory jsou vlastně 2D filtry typu dolní propust. Při jejich použití však dochází i ke ztrátě ostrosti obrazu. Proto je třeba hledat kompromis mezi potlačením šumu a rozostřením obrazu. To je možné provádět změnou velikosti masky nebo změnou koeficientů masky. Zvýšením hodnoty prostředního prvku masky dojde k snížení rozostření obrazu ale na úkor odstranění šumu.[7]

Příklad průměrujících operátorů:

$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad h = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Pro odstranění šumu typu pepř a sůl se používá mediánový filtr. Ten seřadí jednotlivé prvky z daného okolí masky podle jejich velikosti a jako výstupní prvek vybere prostřední hodnotu. Jedná se o nelineární operátor.[7]

Operátory pro zostřování a detektory hran

Oproti operátorům pro potlačení šumu je nyní kladen požadavek na zvýšení podílu vyšších prostorových frekvencí obsahujících informace o hranách v daném obraze. Spolu se zvýrazněním hran dochází i k zvýraznění šumu. Proto je opět nutné volit kompromis.[7]

Pro zvýraznění hran v obraze se nejčastěji používají, detektory hran založené na hledání maxim prvních derivací (Robertsův, Prewittov, Kirschův a Sobelův) a detektory založené na detekci průchodu druhých derivací nulou (Laplacián).

Příklad Sobelova operátoru pro detekci vodorovných a svislých hran:

$$h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad h = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Příklad operátoru Laplacián:

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

5 LINEÁRNÍ ČÍSLICOVÉ FILTRY

5.1 Úvod

Lineární číslicové filtry jsou algoritmy sloužící k úpravě diskrétních signálů tak, aby došlo k potlačení případně zvýraznění určité složky signálu, který vstupuje do číslicového filtru. Algoritmy číslicových filtrů využívají operací součtu, součinu a zpoždění. Vlastní číslicový filtr je popsán lineární diferenční rovnicí s konstantními koeficienty v časové oblasti nebo přenosovou funkcí v z-rovině. Základní rozdělení lineárních číslicových filtrů je podle jejich impulsní charakteristiky, na filtry s konečnou impulsní charakteristikou FIR (Finite Impulse Response) a na filtry s nekonečnou impulsní charakteristikou IIR (Infinite Impulse Response).[8]

5.2 Číslicové filtry s konečnou impulsní charakteristikou (FIR)

Číslicové filtry s konečnou impulsní FIR (Finite Impulse Response) jsou takové filtry, jejíž hodnoty impulzní charakteristiky jsou současně koeficienty filtru. Diferenční rovnice tohoto filtru je dána podle vztahu (2), tato rovnice současně reprezentuje přímou realizaci FIR filtru. [9][7]

$$y[n] = \sum_{k=0}^{N-1} x[n-k] \cdot a_k \qquad (2)$$

Kde N přestavuje řád číslicového filtru, a_k jsou koeficienty filtru a současně hodnoty impulzní charakteristiky, x[n] přestavuje vstupující vzorky signálu do filtru a y[n] výstupní vzorky zpracovaného signálu filtrem. Z diferenční rovnice (2) lze pomoci *Z* transformace vyjádřit přenos filtru.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{n=0}^{N-1} a_n z^n}{z^{N-1}} \qquad (3)$$

Z přenosu FIR filtru je vidět že všechny jeho póly leží uprostřed jednotkové kružnice, to znamená že, filtr typu FIR je vždy stabilní. Této vlastnosti se často využívá při adaptivní filtraci, při které pak není nutné vyhodnocovat stabilitu navrženého filtru.[9]

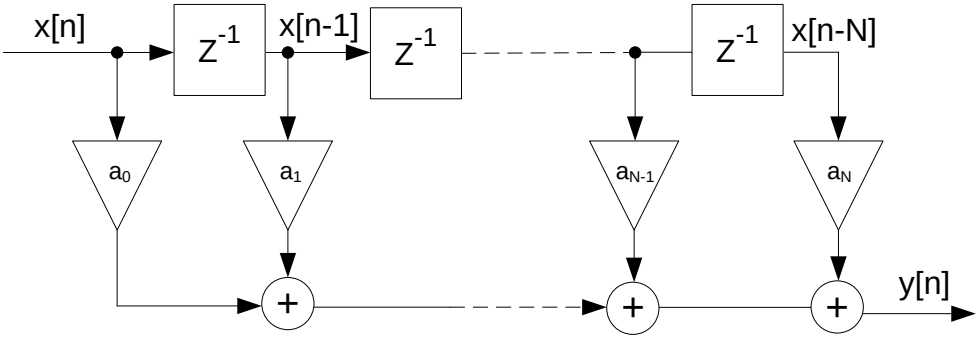
Jednou z dalších výhod FIR filtrů je možnost dosáhnou lineární fázové charakteristiky na celém frekvenčním rozsahu. To znamená, že skupinové zpoždění, které se vypočte jako záporná derivace fáze podle úhlové frekvence je konstantní. Všechny složky signálu vstupujícího do filtru jsou pak na výstupu za stejný časový

interval. Podmínkou pro dosažení lineární fázové charakteristiky je symetrická nebo antisymetrická impulzní charakteristika FIR filtru.[9]

Značnou nevýhodou FIR filtrů je vysoký řád, při požadavku na velkou strmost filtru nebo požadavku na úzké pásmo. Vysoký řád filtru se projeví ve zvýšení výpočetní náročnosti a zvýšením paměťové náročnosti, ale také prodloužení přechodných dějů a zpožděním vstupujícího signálu.[9]

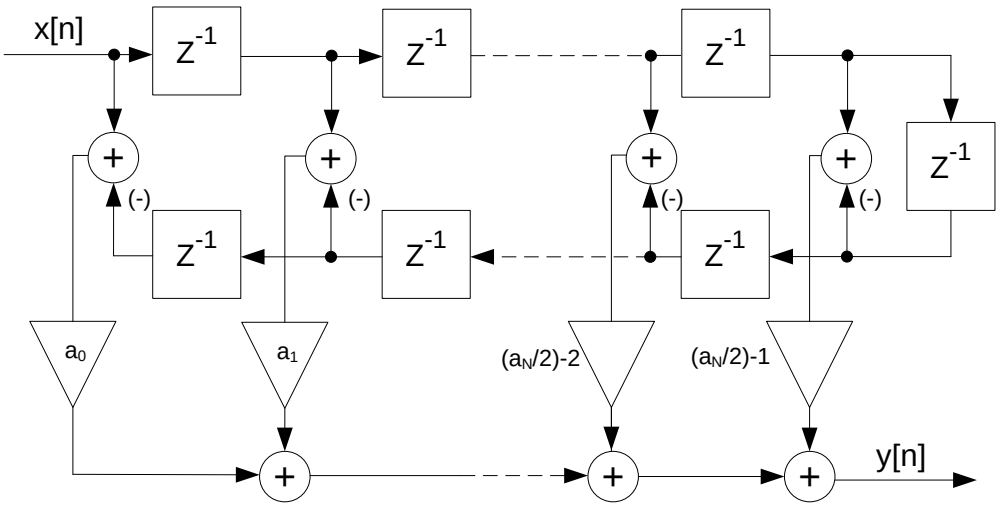
5.3 Základní struktury číslicových filtrů FIR

Nejčastěji používaná struktura je přímá někdy také označovaná jako Transverzální, která vychází přímo z diferenční rovnice (2), struktura je zobrazená na obrázku 5.1. K této struktuře existuje ještě duální struktura, u které je vstupní signál nejprve násoben váhovými koeficienty a teprve pak zpožďován. U číslicových filtrů tytu FIR z důvodu absence zpětné vazby nezáleží tak moc na volbě struktury jak u filtru typu IIR.[10][11]

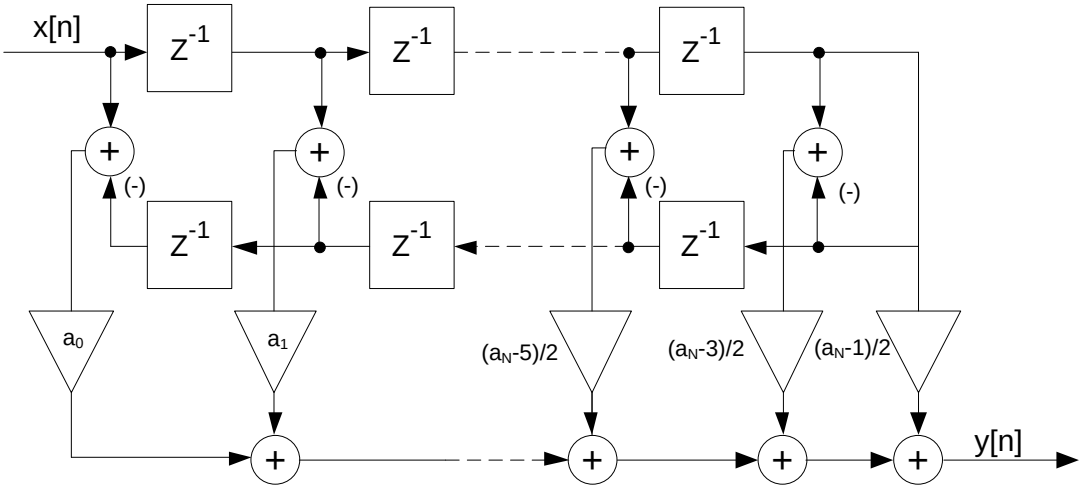


Obrázek 5.1: Přímá struktura FIR filtru

Pokud má filtr lineární fází tedy symetrickou nebo antisymetrickou impulzní charakteristiku lze přímá struktura filtru zjednodušit. Tato zjednodušená struktura je zobrazena na obrázku 5.2 pro sudý řád a na obrázku 5.3 pro lichý řád filtru N. V případě antisymetrické impulzní charakteristiky jsou signály spodní větve filtru přičítány s opačným znaménkem, jak je naznačeno na obrázcích 5.2 a 5.3. Použití této struktury se sníží počet násobení na polovinu, což je výhodné při implementaci v hradlovém poli nebo signálovém procesoru.[7]



Obrázek 5.2: Struktura pro lineární fázovou charakteristiku pro sudé N



Obrázek 5.3: Struktura pro lineární fázovou charakteristiku pro liché N

5.4 Filtry s nekonečnou impulsní charakteristikou (IIR)

Číslicové filtry s nekonečnou impulsní charakteristikou IIR (Infinite Impulse Response) jsou rekurzivní, obsahují tedy vždy zpětné vazby. Nemusí být proto vždy stabilní jak je tomu u filtrů typu FIR a je vždy po návrhu potřeba ověřit zda všechy póly leží uvnitř jednotkové kružnice. Aby bylo možné filtr realizovat, musí být kauzální, to je splněno, pokud v přenosu (5) je řád polynomu jmenovatele větší nebo rovno řádu čitatele.[9][7]

$$y[n] = \sum_{k=0}^{M} x[n-k] \cdot b_k - \sum_{k=1}^{N} y[n-k] \cdot a_k \qquad (4)$$

Číslicový filtr IIR je popsán diferenční rovnicí (4), v které b_k představují koeficienty dopředných vazeb a a_k koeficienty zpětných vazeb, x[n] jsou vstupující vzorky signálu do filtru a y[n] vzorky výstupního signálu.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{n=0}^{M} b_n z^{N-n}}{\sum_{n=0}^{N} a_n z^{N-n}} \qquad M \leq N \qquad (5)$$

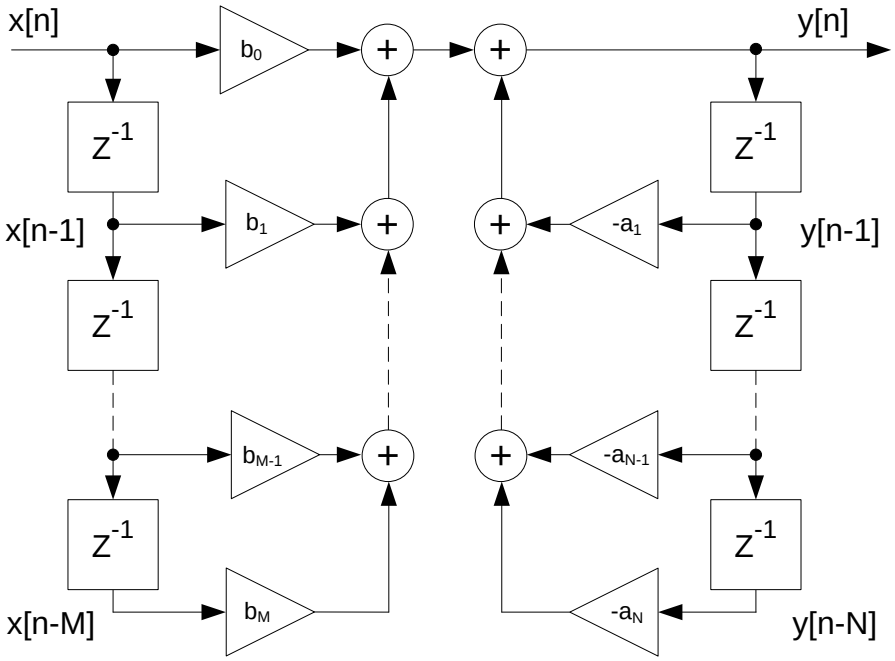
Nevýhodou IIR filtru oproti FIR je nelineární fázová charakteristika, k lineárnímu průběhu fázové charakteristiky je možno se přiblížit ale pouze v malém kmitočtovém rozsahu. Také jsou více náchylné na nepřesnosti, které vznikají při číslicové realizaci a zaokrouhlením koeficientů.[7][11]

Zásadní výhodou IIR filtru je možnost dosáhnutí stejných požadavků jako u FIR filtru s mnohem menším řádem, v literatuře [10] je uváděno, že filtry IIR dosahují přibližně 10krát menšího řádu než filtry FIR při stejném zadání. Z toho plynou menší nároky na paměť, výpočetní náročnost ale také menším zpoždění vstupního signálu a rychlejší přechodný děj. Další odlišností filtrů IIR je že lze nalézt jejich analogový ekvivalent, kdežto filtry FIR existují pouze v číslicové podobě.[7][11]

5.5 Základní struktury číslicových filtrů IIR

5.5.1 Přímá struktura číslicového filtru IIR

Přímá struktura IIR filtru vychází přímo z diferenční rovnice (4). Struktura obsahuje dvojnásobný počet zpožďovacích členů než je řád filtru. Zpožďovací členy jsou zvláště pro vstupní vzorky a zvláště pro výstupní vzorky tak jak je znázorněno na obrázku 5.4. Tato struktura právě díky vyššímu počtu zpožďovacích členů klade vyšší nároky na paměť. Další její nevýhodou je velká citlivost na chyby vznikající zaokrouhlováním. Z těchto důvodů realizace přímé struktury není příliš vhodná.[4][12][13]



Obrázek 5.4: Přímá struktura IIR filtru

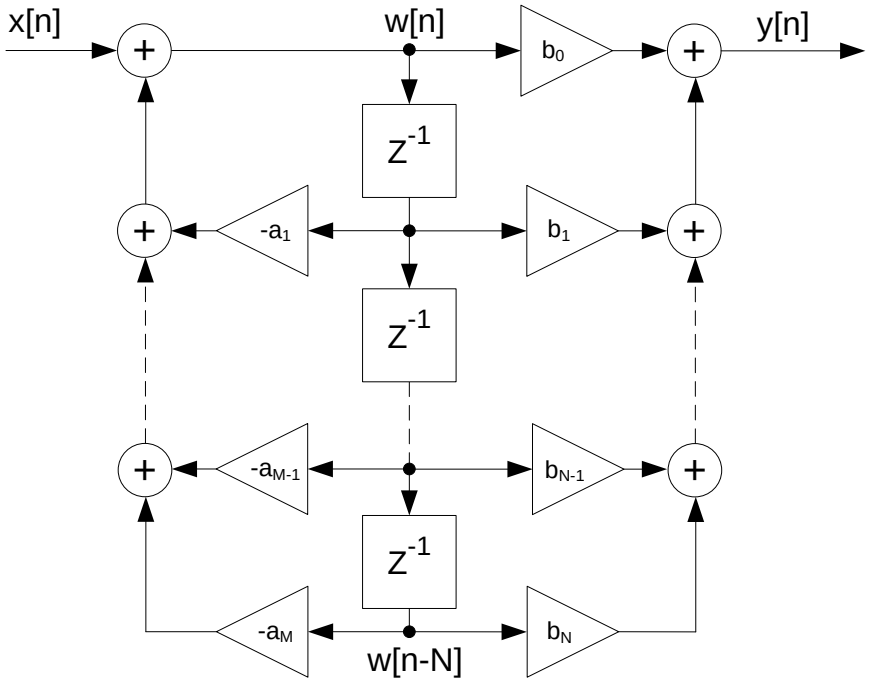
5.5.2 Druhá kanonická struktura číslicového IIR filtru

Druhá kanonická struktura filtru IIR vznikne prohozením pravé a levé strany přímé struktury a sloučením zpožďovacích členů, do kterých po přehození stran přímé struktury vstupují stejné vzorky signálu, tato struktura je zobrazena na obrázku 5.5. Sloučením zpožďovacích členů však dojde ke změně diferenční rovnice, která má nyní dvě části. První část diferenční rovnice (6) popisuje vztah mezi vnitřními stavy a vstupním signálem a druhá část diferenční rovnice (7) popisuje vztah mezi výstupním signálem a vnitřními stavy filtru.[13]

$$w[n] = x[n] - \sum_{k=1}^{M} w[n-k] \cdot a_k \tag{6}$$

$$y[n] = \sum_{k=0}^{M} w[n-k] \cdot b_k \tag{7}$$

Druhá kanonická struktura obsahuje stejný počet zpožďovacích členů jako je řád filtru, stejně tak jako ostatní kanonické struktury. To vede k menším paměťovým nárokům. Oproti přímé struktuře druhá kanonická struktura většinou dosahuje lepších numerických vlastností.[7][13]



Obrázek 5.5: Druhá kanonická struktura

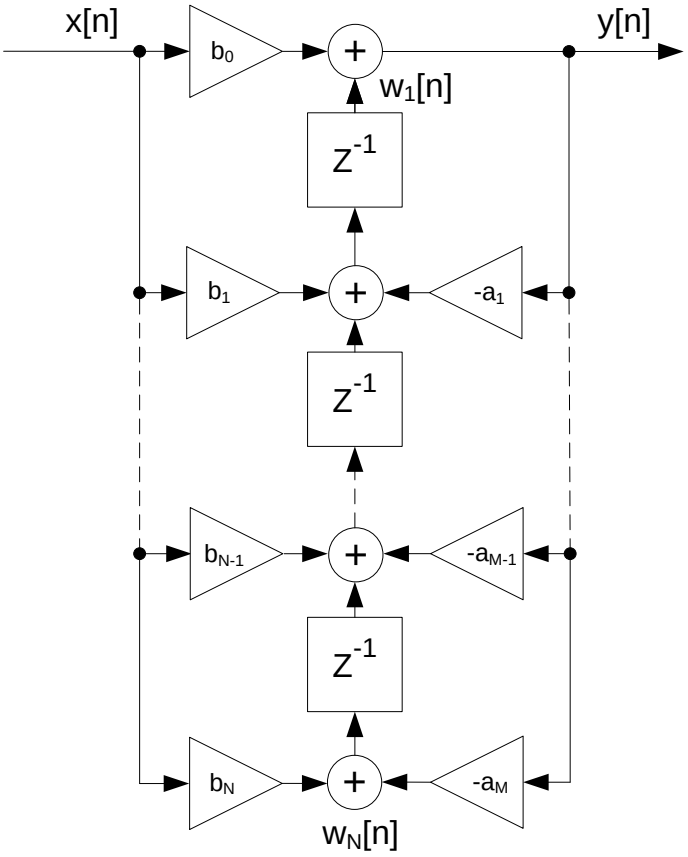
5.5.3 První kanonická struktura číslicového filtru IIR

Otočením směrů signálů druhé kanonické struktury, vznikne první kanonická struktura, zobrazena na obrázku 5.6. Touto úpravou opět dojde ke změně diferenčních rovnic. První diferenční rovnice (8) popisuje výpočet vlastního výstupního signálu, druhá diferenční rovnice (9) popisuje výpočet vnitřních stavů ze vstupního, výstupního signálu a předchozího vnitřního stavu pro k=1 až po k=M-1. Třetí diferenční rovnice (10) představuje poslední součet vstupních a výstupních vzorků signálu. První kanonická struktura má stejně jako ostatní kanonické struktury stejný počet zpožďovacích členů jako je řád filtru a většinou dosahuje dobré odolnosti vůči chybě způsobené kvantováním.[13]

$$y[n] = b_0 x[n] + w_1[n-1] \tag{8}$$

$$w_k[n] = b_k x[n] + w_{k+1}[n] - a_k y[n] \tag{9}$$

$$w_M[n] = b_M x[n] - a_M y[n] \tag{10}$$



Obrázek 5.6: První kanonická struktura

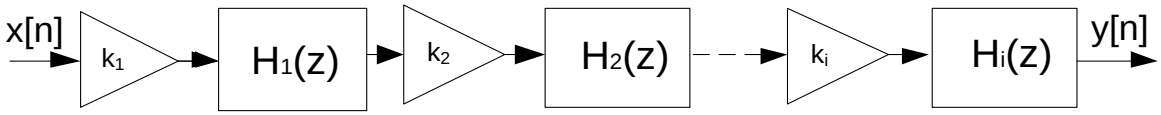
5.5.4 Kaskádní a paralelní struktura

Struktury uvedené výše jsou při vyšších řádech číslicového filtru značně citlivé na zaokrouhlovací chybu. Tato citlivost se dá snížit rozložením filtru na několik sekcí nižšího řádu, které jsou spojeny tak, aby dohromady daly požadovaný přenos. Spojení dílčích sekcí je sériové (kaskádní) nebo paralelní.[7]

Kaskádní struktura

Kaskádní struktura vznikne rozložením číslicového filtru na sekce druhého řádu, které se sériově spojí. Tato struktura je zobrazena na obrázku 5.7, obrázek je navíc doplněn o váhové koeficienty *k,* jejich význam je popsán v následující podkapitole. Jednotlivé sekce druhého řádu lze realizovat kteroukoliv z výše uvedených struktur. Z hlediska úspornosti paměťového místa je však vhodné použít kanonickou strukturu. V literatuře [10] je uváděna první kanonická struktura jako nejvhodnější pro praktickou realizaci z důvodu rovnoměrného rozložení aritmetických operací diferenčních rovnic. Při rozkladu na sekce druhého řádu se vychází ze základního vztahu pro přenos, pro IIR filtry to je (5). Tento přenos se rozloží na součin racionálních lomených funkcí druhého řádu. Výsledný přenos je pak dán jako součin jednotlivých přenosů sekcí druhého řádu.

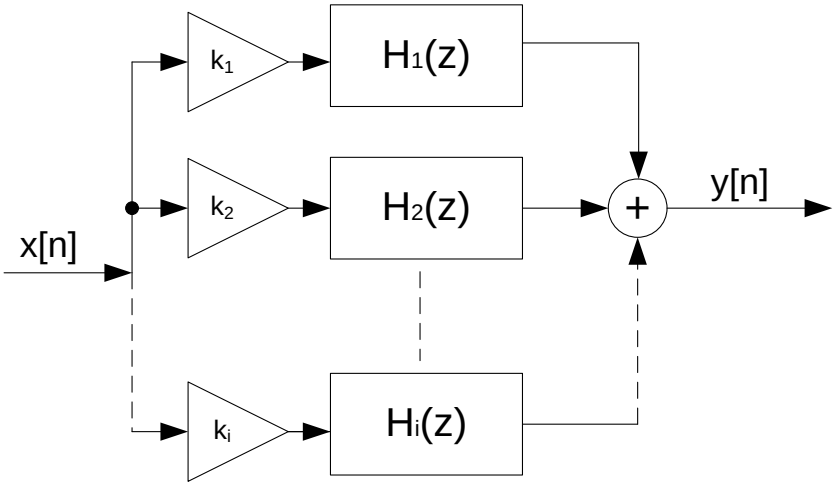
Jednou z výhod kaskádní struktury je možnost při dostatečně výkonném obvodu využívat opakovaně pouze jednu sekci, které jsou měněny koeficienty a vnitřní stavy.[7][10]



Obrázek 5.7: Kaskádní struktura filtru

Paralelní struktura

Paralelní struktura podobně jako kaskádní může mít jednotlivé sekce realizované libovolnou strukturou. Opět je však vhodné použít kanonickou strukturu. Koeficienty pro jednotlivé sekce se získají rozkladem přenosu daného v základním tvaru (5) na součet parciálních zlomků. Výhoda této struktury je především při hardwarové realizaci, u které je možnost zajistit fungování všech sekcí současně. To vede k menšímu zpoždění vstupního signálu.[7]



Obrázek 5.8: Paralelní struktura filtru

Váhové koeficienty

Váhové koeficienty *k*, zobrazené na obrázcích 5.7 a 5.8 mají význam pouze při implementaci dané struktury filtru v hradlovém poli, mikroprocesoru či signálovém procesoru, které používají aritmetiku s pevnou desetinou čárkou. Při použití aritmetiky s pevnou desetinou čárkou mohou nastat problémy s přetečením nejvýznamnějšího bitu při matematických operacích. Hodnoty, kterých mohou čísla v algoritmu, který používá aritmetiku s pevnou desetinou čárkou dosáhnout bývají zpravidla upraveny tak, aby se

nacházely v intervalu ±1. Tato úprava současně zaručuje, aby při násobení dvou čísel byl výsledek opět v rozsahu ±1. Jediný případ, kdy může tedy dojít k přetečení nejvýznamnějšího bytu je při sčítání. Tomu se právě snaží zabránit váhové koeficienty, kterými se násobí signál vstupující do jednotlivých sekcí. Přitom hodnota váhového koeficientu je menší jak jedna. Tím je dosaženo snížení úrovně signálu na takovou hodnotu, při které by nemělo dojít v dané sekci při operaci sčítání k přetečení nejvýznamnějšího bytu.[10]

6 NÁVRH ČÍSLICOVÝCH FILTRŮ PRO IMPLEMENTACI V FPGA

6.1 Parametry navrhovaných číslicových filtrů FIR a IIR

Zvolené parametry FIR filtru

Pro návrh číslicového filtru s konečnou impulsní charakteristikou FIR byla použita metoda váhových oken, která je jednou z nejjednodušších metod pro návrh FIR filtru. Při návrhu bylo využito programu Matlab, který touto metodou filtr navrhl pro požadované parametry.

Navrhovaný filtr je typu horní propust se zlomovou frekvencí 10 kHz. Řád filtru byl zvolen patnáctého řádu s ohledem na množství koeficientů a požadovaný sklon. Vzorkovací frekvence, pro kterou je filtr navržen, musí být zvolena tak, aby byl splněn Shannon-Kotelníkův teorém. To znamená, že vzorkovací frekvence musí být alespoň větší jak dvojnásobek nejvyšší harmonické složky obsažené ve spektru signálu, vstupujícího do filtru. V našem případě předpokládáme, že vstupní signál do filtru bude audio signál, proto je vzorkovací frekvence zvolena o kmitočtu 44 kHz. Váhovací okno, které se při použití metody váhových oken volí a které ovlivňuje tvar frekvenční charakteristiky filtru, bylo zvoleno Hammingovo okno.

Zvolené parametry IIR filtru

Číslicový filtr s nekonečnou impulsní charakteristikou IIR byl stejně jako filtr FIR navrhnut za pomocí použití programu Matlab. Navrhovaný IIR filtr je typu Butterworth, který je charakteristický plochou amplitudovou charakteristikou a současně tím i nulovým překmitem. Stejně jako u FIR filtru se jedná o horní propust se zlomovou frekvencí 10 kHz. Řád filtru byl zvolen osmého řádu. Vzorkovací frekvence, pro kterou je filtr navržen je stejná jako u FIR filtru tedy 44 kHz.

6.2 Převod koeficientů filtrů do fixed-point tvaru

Způsob uložení reálných čísel v paměti nebo registrech může být ve tvaru s plovoucí řádovou čárkou floating-point nebo s pevnou řádovou čárkou fixed-point. Číslo uložené ve tvaru s plovoucí řádovou čárkou, se skládá ze znaménkového bitu, exponentu a mantisy. Přitom znaménkový bit je prvním bitem takto reprezentovaného čísla, za ním se nachází exponent, v kterém je uložena informace o poloze řádové čárky, poslední část je mantisa, která nese informaci o vlastním čísle. Při reprezentaci čísla ve formátu s pevnou řádovou čárkou je poloha řádové čárky pevně stanovena programátorem a informace o poloze řádové čárky už není přenášena.[14]

Použití reprezentace čísel ve formátu s pevnou řádovou čárkou má především výhodu v jednouché implementaci základních matematických operací, s kterými souvisí i rychlost jakou jsou operace prováděny. Této výhody se především využívá u mikroprocesorů, signálových procesorů a hradlových polí, u kterých nemusí být obsažen matematický koprocesor pro vykonávání operací s plovoucí řádovou čárkou.[14]

6.2.1 Reprezentace koeficientů filtrů ve fixed-point tvaru

Při reprezentaci koeficientů filtru ve formátu s pevnou desetinou čárkou je vhodné, aby všechny koeficienty i vstupní signál nabývaly hodnotu v intervalu <-1,1>. Tím se zajistí, že při násobení dvou čísel nemůže dojít k přetečení nejvýznamnějšího bitu. Jedinou operací, při které může dojít k přetečení nejvýznamnějšího bitu je součet. Proto je potřeba po převedení koeficientů do formátu s pevnou řádovou čárkou vždy ověřit, zda vlivem sčítání nedochází v některých místech filtru k přetečení. Pokud k přetečení dochází tak jednou z možností jak tomu zabránit je snížit úroveň signálu, ještě před tím než vstoupí do filtru a na výstupu filtru jej opět o stejnou hodnotu zvýšit. Snížením vstupního signálu však dochází i k snížení přesnosti, proto se tento postup dá použít jen v omezené míře. Další možností jak omezit přetečení nejvýznamnějšího bitu při součtu je volba struktury filtru, kdy například u kaskádní a paralelní struktury může být signál váhován před každou sekcí.

Problematika reprezentace koeficientů u IIR filtrů

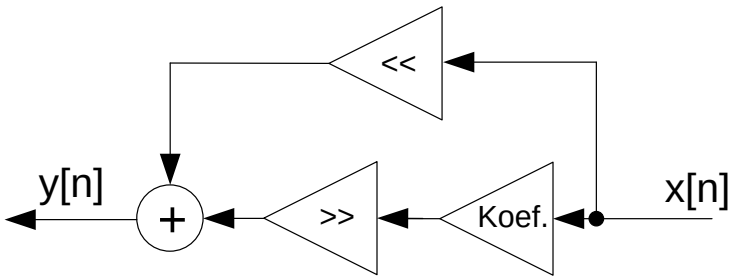
Při realizaci IIR filtru pomocí struktur, které nevyužívají rozdělení do sekcí menších řádů je zcela běžné, že koeficienty čitatele nabývají řádově nižších hodnot než jedna. Oproti tomu koeficienty jmenovatele jsou řádově větších hodnot než jedna. Tím se však komplikuje uložení čísel ve formátu s pevnou řádovou čárkou. Řešení těchto problémů mohou být následující.

Pokud jsou koeficienty čitatele přenosu řádově nižších hodnot, doporučuje se vynásobit je dvěma. Výstupní signál takto upraveného filtru musí být však snížen o polovinu. Touto úpravou se zmenší vzdálenost mezi řádovou čárkou a první aktivním bitem a tím se zvětší přesnost uložených koeficientů. [15]

V případě, že koeficienty jmenovatele jsou řádově větší než jedna, je zapotřebí značné množství bitů pro reprezentaci části umístěné před řádovou čárkou. Tyto bity pak mohou chybět při reprezentování části za řádovou čárkou. Tím dojde k výraznému snížení přesnosti koeficientů, což může vést až k nestabilitě filtru. Řešením je použití většího množství bitů pro reprezentaci koeficientů. Tím však dojde i k nárůstu použitých násobiček v poli.[15]

Jednou z dalších možností je zmenšit část před řádovou čárkou tak aby pro její reprezentaci bylo potřeba co nejmenší počet bitů. To lze provést rozdělením koeficientu na dvě části. Například pokud v ideálním případě vyjde koeficient o hodnotě 32,3291,

můžeme jej rozdělit na část 32, kterou lze realizovat pomocí bitového posuvu vstupního signálu vlevo. Druhou část 0,3291 lze již klasicky uložit ve formátu s pevnou řádovou čárkou. V případě, že koeficient je reprezentován šestnácti bity a vstupní signál je také šestnácti bitový, tak výstup z násobičky je třiceti dvou bitový. Bitový posun doprava pak musí být patnáct bitů. Oba signály, které vstupují do sčítačky, jsou pak šestnácti bitové. Nevýhodou tohoto řešení je použití sčítačky navíc. Tím však dojde k navýšení doby výpočtu o operaci součtu. Na obrázku 6.1 je zjednodušeně blokově zobrazen popsaný postup.[15]

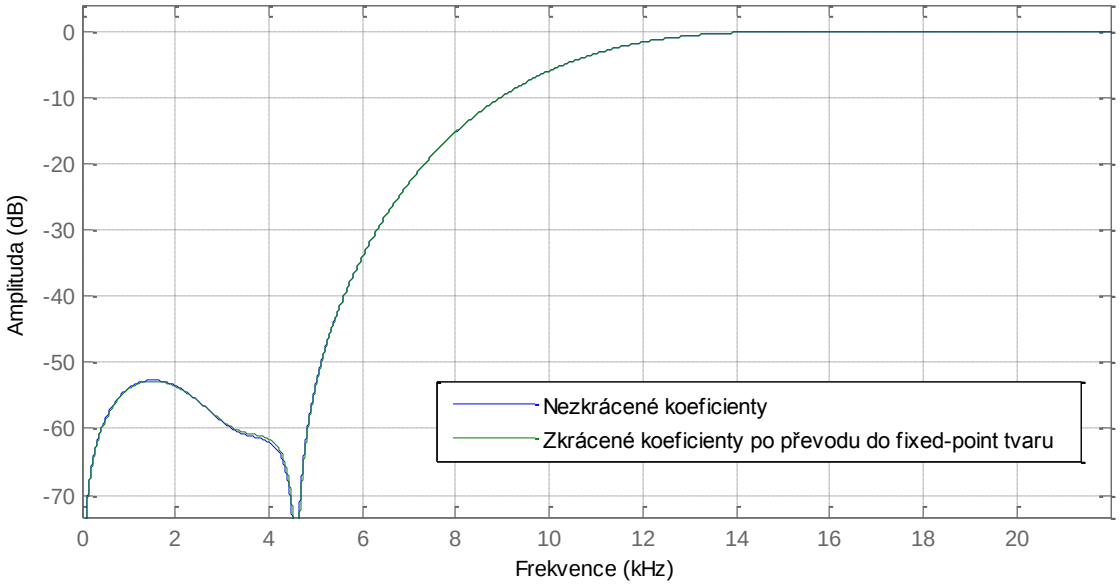


Obrázek 6.1: Blokové schéma pro uložení velkého čísla s desetinou čárkou v FPGA

6.2.2 Koeficienty u navrženého FIR filtru

U FIR filtru hodnoty koeficientů vždy vycházejí v intervalu <-1,1>, to umožňuje ideální převedení do tvaru s pevnou řádovou čárkou bez nutnosti dalších úprav.

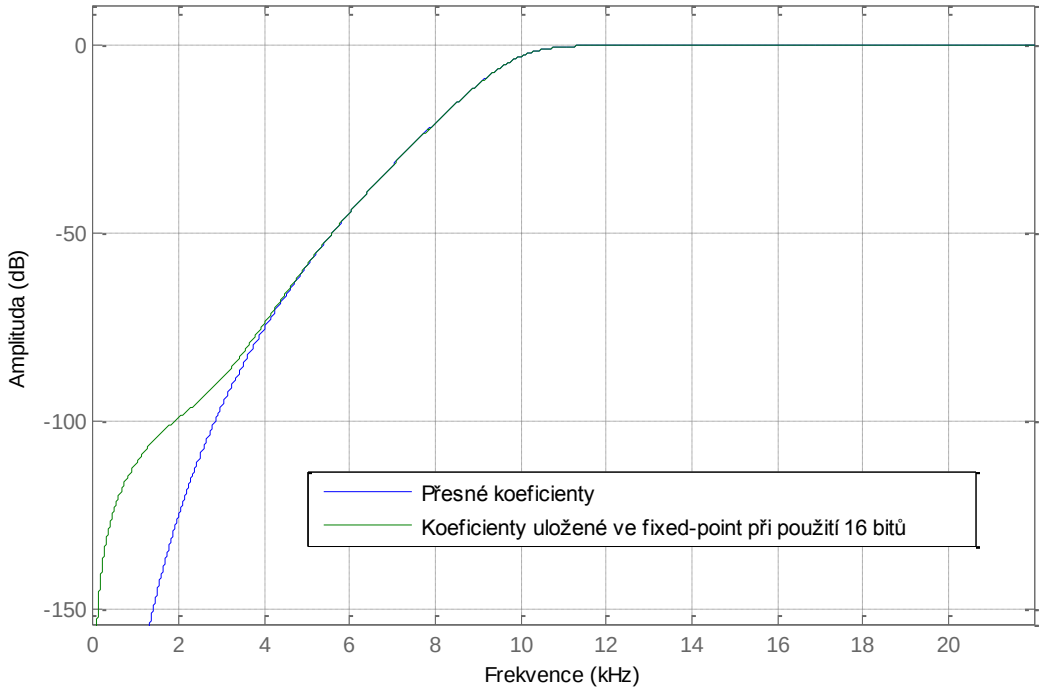
V našem případě bylo pro uložení koeficientů ve formátu s pevnou desetinou čárkou použito šestnácti bitů, z toho první bit je znaménkový. Při převedení navržených koeficientů filtru do tvaru s pevnou desetinou čárkou dojde ke snížení jejich přesnosti, což se projeví ve změně kmitočtové a fázové charakteristiky. Srovnání kmitočtové charakteristiky filtru s původními koeficienty a koeficienty převedenými do fixed-point tvaru je zobrazeno na obrázku 6.2.



Obrázek 6.2: Srovnání frekvenčních charakteristiky FIR filtru

6.2.3 Koeficientu u IIR filtrů

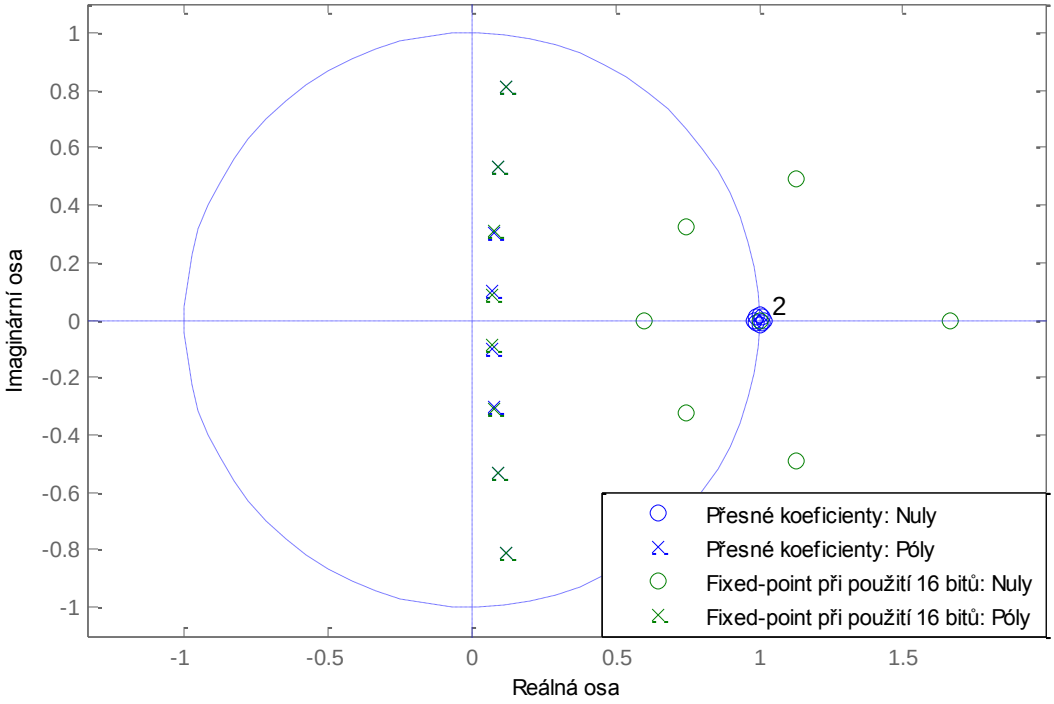
V mém případě koeficienty navrženého filtru pro struktury, které nevyužívají rozdělení do sekcí menších řádů, vyšly o hodnotách nepřesahující interval <-2,2>. V takovém případě není nutné provádět úpravy popsané v předchozí kapitole. Stačí pouze koeficienty čitatele a jmenovatele přenosové funkce podělit dvěma. Tato úprava se projeví koeficienty o poloviční hodnotě ve struktuře filtru, kromě koeficientu a_0 který je dvojnásobný. Tento koeficient je zpravidla roven jedné, po úpravě má tedy hodnotu dva. Násobení dvěma se v aritmetice s pevnou řádovou čárkou dá výhodně realizovat jako bitový posun o jeden bit doprava.



Obrázek 6.3: Srovnání frekvenčních charakteristik IIR filtru

Pro takto upravené koeficienty bylo použito šestnácti bitů pro uložení ve tvaru s pevnou řádovou čárkou. Při převedení koeficientů filtru do tvaru s pevnou desetinou dojde k snížení jejich přesnosti, což se projeví ve změně kmitočtové a fázové charakteristiky. Na obrázku 6.3 je vidět jaký má vliv snížení přesnosti koeficientů na frekvenční charakteristiku. Viditelná změna na frekvenční charakteristice se projevuje až pod -50dB.

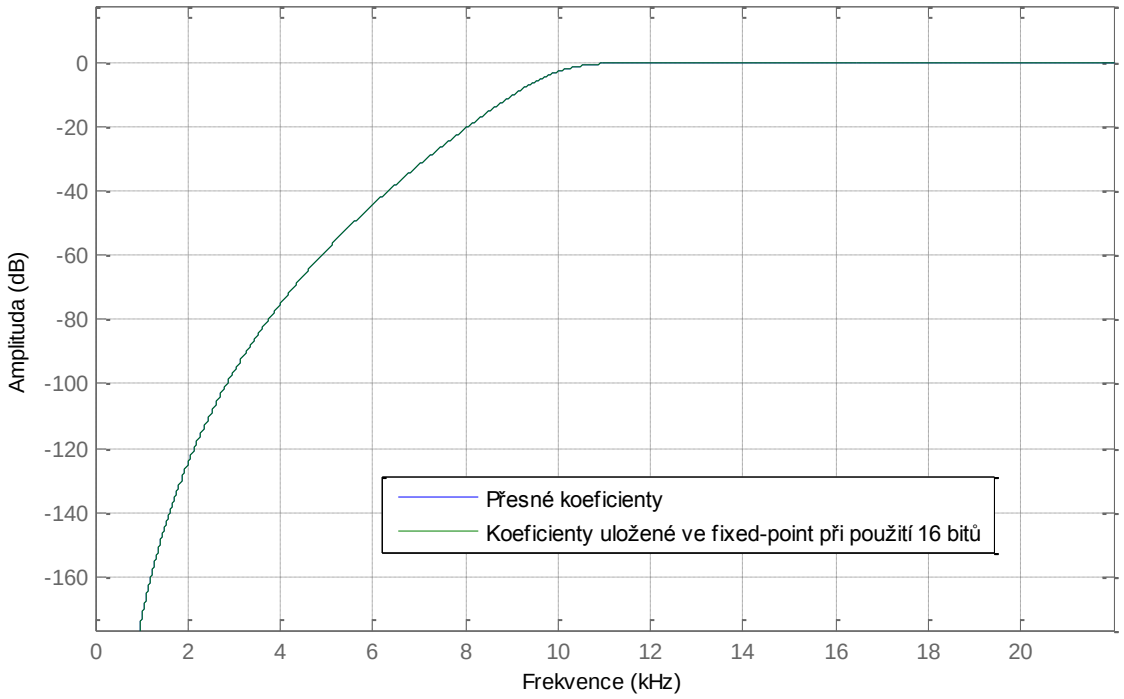
U filtru typu IIR může dojít vlivem snížení přesnosti koeficientů k posunu pólů mimo jednotkovou kružnici a tím k nestabilitě filtru. Z toho důvodu se musí vždy zkontrolovat, jestli po převedení do formátu s pevnou řádovou čárkou všechny póly leží uvnitř jednotkové kružnice. Na obrázku 6.4 je zobrazena jednotková kružnice. Jak je vidět, tak vlivem zaokrouhlením došlo jen k nepatrnému posunutí pólů a žádný neopustil jednotkovou kružnici, filtr je tedy stabilní a může být realizován.



Obrázek 6.4: Srovnání jednotkové kružnice IIR filtru

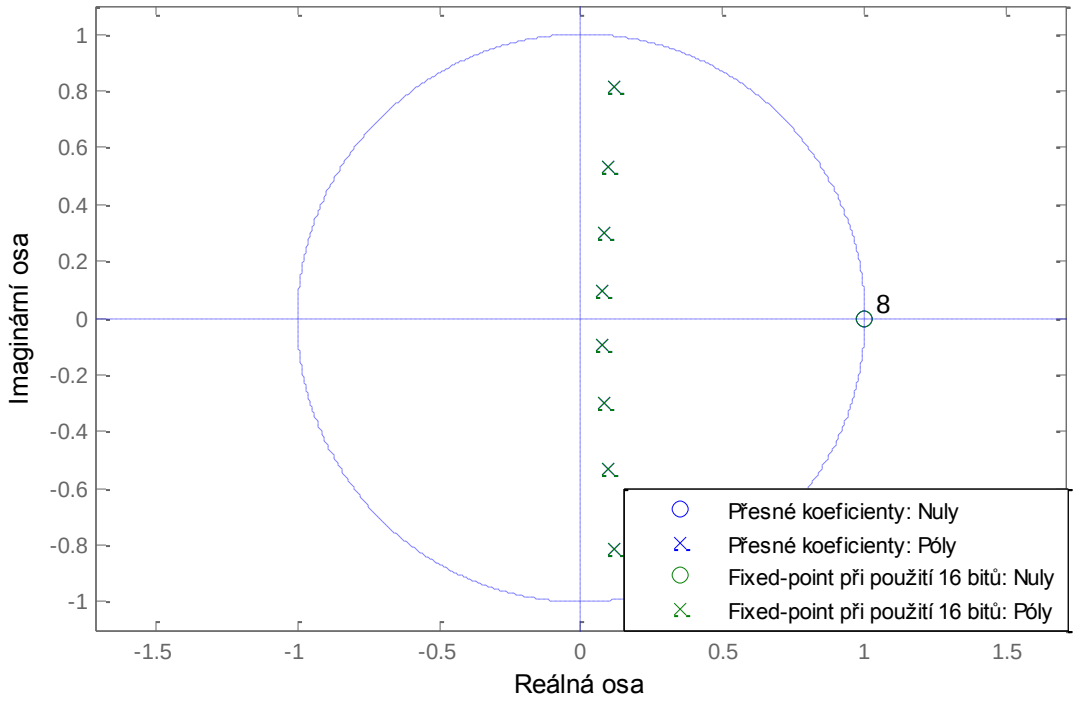
6.2.4 Koeficienty u kaskádní struktury IIR filtrů

Při použití kaskádní struktury filtru zobrazené na obrázku 5.7, se koeficienty vždy nacházejí v intervalu <-2,2>. V takové případě stačí pouze koeficienty čitatele a jmenovatele přenosové funkce podělit dvěma. Tím se dosáhne, aby koeficienty nabývaly hodnot pouze v intervalu <-1,1>. Tento postup byl již podrobně popsán v předchozí kapitole. Pro uložení koeficientů ve tvaru s pevnou řádovou čárkou bylo použito šestnácti bitů stejně jako u předchozích struktur. Na obrázku 6.5 jsou zobrazeny frekvenční charakteristiky pro původní koeficienty a koeficienty po převodu do šestnácti bitového vyjádření s pevnou řádovou čárkou. Je vidět, že snížením přesnosti koeficientů nedošlo prakticky k žádné viditelné změně na frekvenční charakteristice.



Obrázek 6.5: Srovnání frekvenčních charakteristik IIR filtru kaskádní struktury

Obrázek 6.6 ukazuje, že snížení přesnosti se neprojevilo ani v rozmístění pólů a nul. Za zmínku stojí, že rozmístění nul u nekaskádních struktur, které nemají sníženou přesnost je v těsné blízkosti bodu [1,0], což ukazuje obrázek 6.4. Zatím co u kaskádní struktury jsou nuly umístěny přesně v bodě [1,0], obrázek 6.6. To je způsobeno značnou citlivostí nekaskádních struktur na přesnost koeficientů, která se projevila i u přímo vypočtených koeficientů. Pro získání koeficientů pro jednotlivé sekce kaskádní struktury bylo z tohoto důvodu v Matlabu využito možnosti přímého získání těchto koeficientů bez nutnosti rozkladu celkového přenosu filtru. Pokud by se však koeficienty pro jednotlivé sekce získaly rozkladem celkového přenosu. Rozložení nul, které nemají sníženou přesnost, by bylo stejné jak na obrázku 6.4.



Obrázek 6.6: Srovnání jednotkové kružnice IIR filtru kaskádní struktury

7 SESTAVENÉ PROCESOROVÉ SYSTÉMY

Před vlastní implementací algoritmů je potřeba nejprve sestavit procesorový systém. Sestavení procesorového systému se provádí v nástroji SOPC Builder. Pro implementaci číslicových lineárních filtrů v hradlovém poli pomocí C2H je sestaven jiný procesorový systém, než pro implementaci hranových operátorů s možností zobrazování zpracovaného obrazu na LCD displeji, který je umístěn na vývojovém kitu. Jednotlivé sestavené procesorové systémy jsou uvedeny v příloze 1 spolu se seznamem všech řídících hodin, které jsou v daném systému použity.

7.1 Použité komponenty

Nios II

Hlavní částí celého systému je soft-core procesor Nios II s třiceti dvou bitovou redukovanou instrukční sadou (RISC) a třiceti dvoubitovou datovou a adresovou sběrnicí. Výhodou tohoto procesoru je značná flexibilita, která umožňuje přizpůsobovat procesor pro konkrétní aplikaci. Jedním z hlavních přizpůsobení je výběr vlastního jádra procesoru. K dispozici jsou tři verze jader. Rychlé jádro Nios II/f určené pro provádění náročných operací, současně také umožňuje provádět nejvíce nastavení, standartní jádro Nios II/s navržené na dostatečný výkon a malou velikost a ekonomické jádro Nios II/e, jedná se o jádro navržené na co nejmenší velikost. V sestavených systémech je použit procesor Nios II/f s rychlým jádrem.[16]

Tristate Bridge

Tristate Bridge umožňuje sdílet datové, adresové a řídící piny. Při použití více externích pamětí je tak možno ušetřit značné množství pinů.

Clok Crossing Bridge

Komponenta umožňuje propojení master a slave portu pracujících na rozdílných hodinách. Pokud komponenta není použita SOPC Builder vygeneruje automaticky logiku pro přechod na jiný řídící hodinoví signál. Tato logika však neumožňuje optimální výkon pro náročnější operace. Základem vlastního Clock Crossing Bridge jsou dvě parametrizovatelné FIFO paměti pro oba směry přenosu dat.[2]

Pipeline Bridge

Pipeline Bridge je komponenta, která vkládá mezi master a slave port registr. Pipeline Bridge tak umožňuje snížit kritické zpoždění mezi master a slave portem a zvýšit maximální možnou frekvenci. Při jejím použití však dochází k zvýšení latence o jeden hodinový cyklus v obou směrech.[2]

System ID

Systém ID poskytuje unikátní identifikátor, podle kterého Nios II IDE ověří, jestli je program vytvořen pro aktuální konfiguraci hradlového pole. Pokud identifikátor nakonfigurovaného FPGA a identifikátor v Nios II IDE nesedí, tak není umožněno nahrání programu do paměti a je poukázáno na rozdílné identifikátory systému. [2]

Ostatní použité komponenty

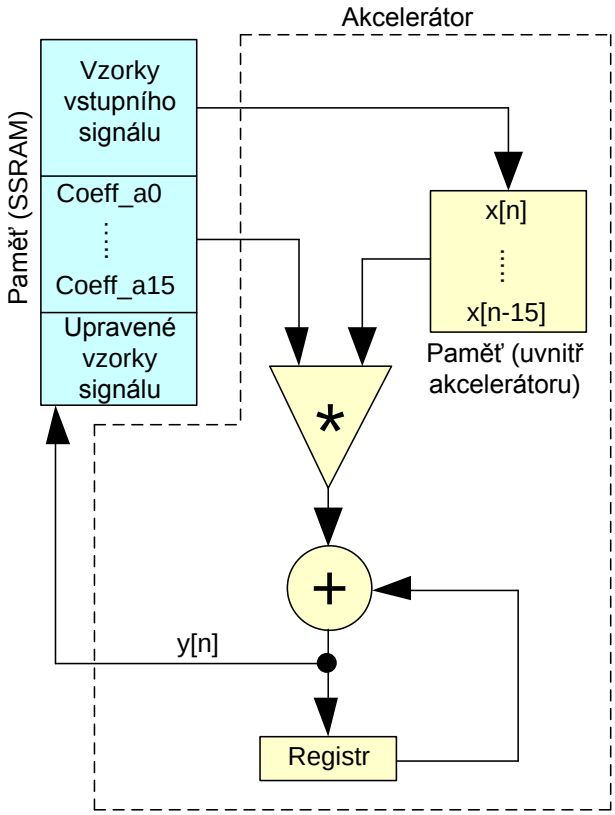
Jednou z dalších použitých komponent je JTAG UART. Tato komponenta umožňuje komunikaci mezi počítačem a vlastním FPGA. Dále je použita komponenta Performance Counter, která slouží pro přesné měření časových intervalů, za které jsou provedeny zvolené částí kódu. Pro vlastní časování je použit Interval Timer. Nastavení PLL, tak aby výstupní hodinové signály odpovídaly zadaným požadavkům, umožňuje komponenta ALTPLL. Pro vytvoření vstupů případně výstupů z hradlového pole slouží PIO (Parallel I/O). Dále jsou použity komponenty realizující řadiče pamětí a LCD displeje. [2]

8 IMPLEMENTACE ČÍSLICOVÝCH FILTRŮ V FPGA POMOCÍ TECHNIKY C2H

Všechny implementované struktury FIR filtrů jsou patnáctého řádu a všechny struktury IIR filtrů osmého řádu. U optimalizovaných realizací jednotlivých filtrů je snaha o co největší výkon na úkor spotřebovaných zdrojů a nárůstu latence.

8.1 Realizace FIR filtru s jednou násobičkou

Následující realizace akcelerátoru FIR filtru vychází z přímé struktury zobrazené na obrázku 5.1. Akcelerátor obsahuje jednu třiceti dvou bitovou násobičku, která provádí násobení koeficientů se zpožděnými vzorky vstupního signálu. Koeficienty filtru a vzorky vstupního signálu jsou uloženy v paměti SSRAM. Z této paměti jsou prostřednictvím Avalon master portu koeficienty přiváděny do násobičky a vzorky vstupního signálu do paměti realizované uvnitř akcelerátoru. Tato paměť slouží pro uložení zpožděných vzorků vstupního signálu, které jsou přiváděny do násobičky. Všechny násobené koeficienty a zpožděné vzorky vstupního signálu jsou postupně sčítány sčítačkou. Výsledný vypočtený vzorek je pak prostřednictvím Avalon master portu uložen do paměti SSRAM. Popsaný postup je blokově naznačen na obrázku 8.1.



Obrázek 8.1: Akcelerátoru FIR filtru s jednou násobičkou

V tomto případě kód pro C2H překladač nebyl nijak optimalizován pro hardwarovou reprezentaci. Jediné rozšíření je o propojovací direktivy pro připojení akcelerátoru k paměti SSRAM a `__restrict__` u ukazatelů.

Vlastní program je složen z jedné hlavní smyčky, která obsahuje dvě vnořené smyčky. Hlavní smyčka zajišťuje postupné spouštění vnořených smyček, dále načtení vzorků vstupního signálu a ukládání výstupních zpracovaných dat do paměti SSRAM. Pro zpožďování vstupních vzorků je v kódu použito pole, tomu v hardwaru odpovídá paměť vytvořená uvnitř akcelerátoru. Tato paměť může být vytvořena z logických elementů nebo z paměťových bloků hradlového pole. To jak bude paměť realizována, rozhoduje kompilátor v závislosti na velikosti požadované paměti a na konkrétním typu použitého hradlového pole. V mém případě je paměť realizovaná pomocí logických elementů. Zpoždění, tedy posunutí dat uložených v této paměti popisuje první vnořená smyčka. Po dokončení této operace je prováděno násobení zpožděných vzorků vstupního signálu s koeficienty filtru a jejich postupné sčítání pro daný vzorek vstupního signálu. Tyto operace popisuje druhá vnořená smyčka.

Takovýto zápis, který je typický pro softwarovou realizaci vytvoří akcelerátor s jednou třiceti dvou bitovou násobičkou a se sedmi Avalon master. Přitom tři Avalon master porty jsou vytvořeny pro ukazatele do paměti SSRAM a zbývající čtyři pro operace s pamětí uvnitř hradlového pole.

	Počet
Násobička (32bitová)	1
Avalon master port	7

Tabulka 8.1: Specifické zdroje pro akcelerátor FIR filtru s jednou násobičkou

V tabulce 8.2 jsou uvedeny všechny využité zdroje při použití pouze procesoru Nios II pro softwarové řešení a při použití procesoru Nios II s hardwarovým akcelerátorem, který realizuje vlastní filtr. V posledním sloupci tabulky je uvedeno o kolik procent se zvýšil nárůst zdrojů při použití hardwarového akcelerátoru.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	5347/24624	22%	+39%
Kombinační funkce (LUT)	3419/24624	14%	4510/24624	18%	+32%
Vyhrazené logické registry	2314/24624	9%	3482/24624	14%	+51%
Registry	2439		3607		+48%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	6/132	5%	+50%
PLL	1/4	25%	1/4	25%	

Tabulka 8.2: Využité zdroje pro FIR filtr s jednou násobičkou

Srovnání rychlosti algoritmu FIR filtru realizovaného softwarově a implementace filtru do hardwaru jako akcelerátoru je uvedena v tabulce 8.3. Takto realizovaný akcelerátor umožňuje 8,3x rychlejší zpracování dat, oproti softwarové realizaci.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	5512,22	
Implementace s použitím akcelerátoru	659,82	8,3x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.3: Srovnání rychlosti pro FIR filtr s jednou násobičkou

V tabulce 8.4 jsou uvedeny maximální frekvence pro hodiny vstupující do celého procesorového systému a pro hodiny, na kterých může běžet samostatný hardwarový akcelerátor. Maximální frekvence je kompilátorem počínána mezi počátečními a cílovými registry, které jsou řízeny stejným hodinovým signálem. Cesty s odlišným řídícím hodinovým signálem generovaným pomocí PLL jsou ignorovány.

F_{max} systému	119,5 MHz
F_{max} akcelerátoru	199,6 MHz

Tabulka 8.4: Maximální frekvence pro FIR filtr s jednou násobičkou

Maximální frekvence procesorového systému je tedy určena pouze pro části, které běží na původních vstupních hodinách. Části řízené hodinami vytvořenými pomocí PLL jsou ignorovány. Maximální frekvence samostatného akcelerátoru je změřena, tak že akcelerátor je vyjmut z vlastního procesorového systému a je pro něj vytvořen top-level, který zajišťuje, aby do všech vstupů akcelerátoru vstupoval proměnný signál a všechny výstupy byly připojeny na výstupní piny top-levelu. Tím se zajistí, aby při kompilaci nedošlo k nežádoucí optimalizaci, která by vedla ke změně vlastního akcelerátoru a tím i k změně měřené maximální frekvence.

Latence hlavní smyčka	15	cyklů
Latence vnořená smyčka 1	8	cyklů
Latence vnořená smyčka 2	13	cyklů
CPLI hlavní smyčka	9	cyklů
CPLI vnořená smyčka 1	5	cyklů
CPLI vnořená smyčka 2	1	cyklů

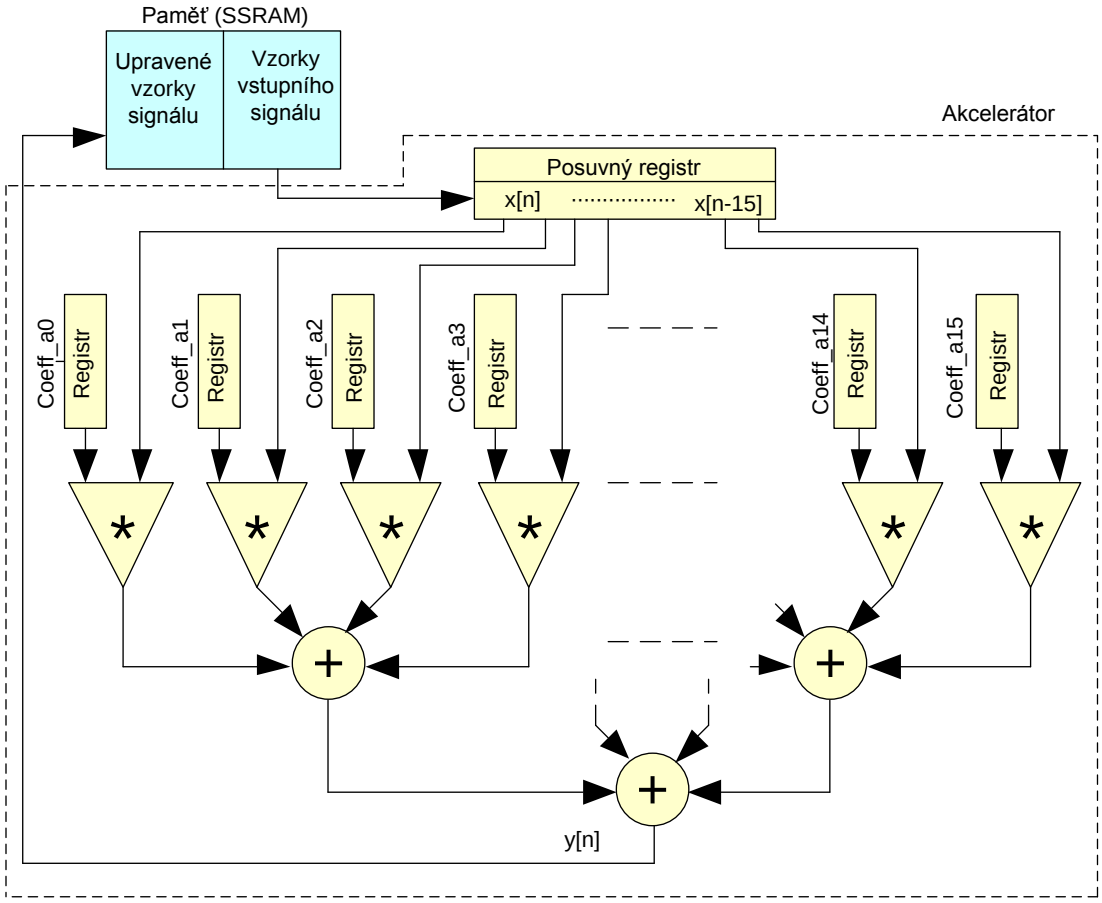
Tabulka 8.5: Latence a CPLI pro FIR filtr s jednou násobičkou

Při neoptimalizovaném řešení pro C2H jsou v kódu použity vnořené smyčky pro které C2H kompilátor vytváří samostatný stavový automat a také pro ně určuje nezávisle na hlavní smyčce latenci a CPLI (Cycles Per Loop Iteration). Latence přitom udává, za kolik hodinových cyklů se vstupní hodnota dostane na výstup, tedy kolik

hodinových cyklů setrvá v dané smyčce, než ji opustí. Oproti tomu CPLI udává počet hodinových cyklů potřebných pro úspěšné dokončení jedné iterace smyčky. V ideálním případě je CPLI rovno jedné, což znamená, že za jeden hodinový cyklus je dosaženo požadované operace.

8.2 Paralelní realizace FIR filtru

Při vhodných optimalizacích kódů pro C2H překladač lze přímou strukturou FIR filtru zobrazenou na obrázku 5.1 realizovat paralelně, tak jak je blokově naznačeno na obrázku 8.2. Oproti předchozí realizaci jsou v paměti SSRAM uloženy pouze vzorky vstupního signálu a vzorky upraveného signálu vystupujícího z akcelerátoru. Zpožďování vstupních vzorků je realizováno posuvným registrem. Koeficienty filtru jsou uloženy přímo v akcelerátoru v registrech vytvořených z logických elementů hradlového pole. Pro každé násobení koeficientu filtru a vzorku vstupního signálu je použita jedna násobička. Pro navržený FIR filtr patnáctého řádu je tedy použito šestnáct násobiček. Výstupy z násobiček jsou sčítány a to tak, že vždy pro čtyři násobičky je použita jedna sčítačka. Všechny tyto mezi součty jsou pak sečteny jednou sčítačkou. Rozdělením sčítání do více kroků se sice zvýší latence akcelerátoru, zato nedojde ke snížení maximální frekvence.



Obrázek 8.2: Paralelní realizace akcelerátoru FIR filtru

Kód pro paralelní realizaci obsahuje jen jednu smyčku bez dalších vnořených, ta zajišťuje, aby byly zpracovány všechny vzorky vstupního signálu, které jsou uloženy v SSRAM paměti.

Vlastní akcelerátor obsahuje oproti předchozím jen dva Avalon master porty, kdy jeden je pro přivádění vstupních vzorků z paměti SSRAM a druhý pro ukládání již zpracovaných vzorků.

	Počet
Násobička (32bitová)	16
Avalon master port	2

Tabulka 8.6: Specifické hardwarové zdroje paralelní realizace akcelerátoru FIR filtru

V tabulce 8.7 je uveden přehled využitých zdrojů. Oproti předchozí realizaci je nárůst využitých logických elementů vyšší o sedm procent. Nejvýznamnější zvýšení je však u použitých devíti bitových vestavěných násobiček, z kterých se vytváří třiceti dvou bitové násobičky.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	5608/24624	23%	+46%
Kombinační funkce (LUT)	3419/24624	14%	4628/24624	19%	+35%
Vyhrazené logické registry	2314/24624	9%	3678/24624	15%	+59%
Registry	2439		3803		+56%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	32/132	24%	+700%
PLL	1/4	25%	1/4	25%	

Tabulka 8.7: Využité zdroje pro paralelní realizaci FIR filtru

Takto realizovaný akcelerátor umožňuje oproti softwarovému řešení, které využívá stejný kód 42,2x, rychlejší zpracování dat. Při srovnání s akcelerátorem, který obsahoval pouze jednu násobičku je rychlost přibližně 18,5x větší.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	1504,53	
Implementace s použitím akcelerátoru	35,66	42,2x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.8: Srovnání rychlosti pro paralelní realizaci FIR filtru

F_{max} systému	130,4 MHz
F_{max} akcelerátoru	192,1 MHz

Tabulka 8.9: Maximální frekvence pro paralelní realizaci FIR filtru

Jak již bylo zmíněno, v kódu nejsou obsažené žádné vnořené smyčky. Spočtená latence a CPLI kompilátorem, tedy odpovídá hardwarovému akcelerátoru. Jak je vidět v tabulce 8.10 CPLI takto realizovaného akcelerátoru je jedna. Akcelerátor tedy teoreticky umožňuje během jednoho hodinového cyklu spočítat jeden výstupní vzorek s latencí 16 hodinových cyklů. Vlastní zrychlení zpracování dat tak především závisí na hodnotě CPLI.

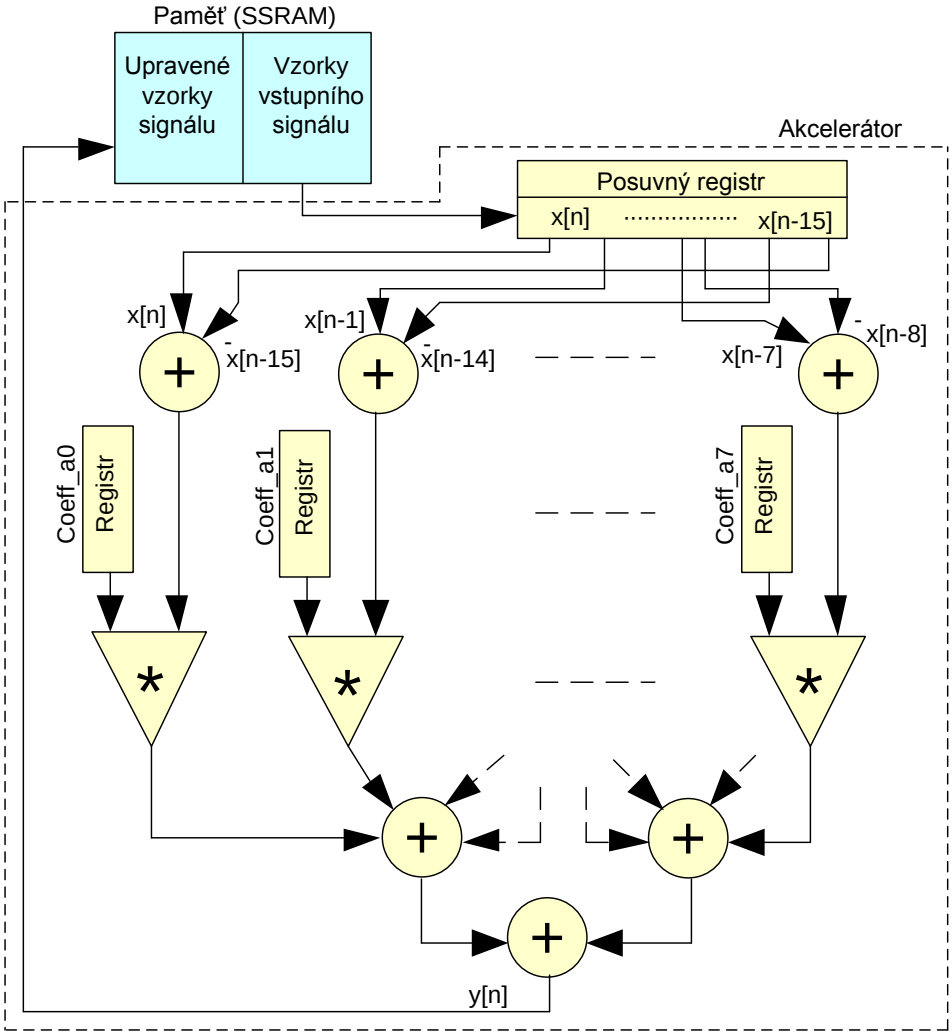
Latence	16	cyklů
CPLI	1	cyklů

Tabulka 8.10: Latence a CPLI pro paralelní realizaci FIR filtru

8.3 Paralelní realizace FIR filtru s poloviční počtem násobiček

Při paralelní realizaci FIR filtru lze počet potřebných násobiček snížit na poloviční počet, pokud se vyjde ze struktur zobrazených na obrázcích 5.2 a 5.3. V tomto případě je navržený filtr patnáctého řádu a má tedy sudý počet koeficientů, proto se vychází ze struktury pro sudý počet koeficientů, která je zobrazena na obrázku 5.3. Současně je také splněn požadavek na antisymetrickou impulsní charakteristiku.

Rozdíl oproti předchozí realizaci je v tom, že vzorky zpožďované posuvným registrem nevstupují přímo do násobiček, ale první polovina vzorků v posuvném registru se z důvodu antisymetrické impulsní charakteristiky odečte od druhé poloviny uložených vzorků. Takto odečtené vzorky pak teprve vstupují do násobičky, kde jsou násobeny koeficienty filtru. Přitom počet uložených koeficientů a tedy i počet použitých registrů je poloviční. Výstupy z násobiček jsou opět sčítány ve více krocích, z důvodu aby nedošlo k snížení maximální frekvence akcelerátoru. Popsaný postup je blokově naznačen na obrázku 8.3.



Obrázek 8.3: Paralelní realizace akcelerátoru FIR filtru s poloviční počten násobiček

V tabulce 8.11 je vidět, že vytvořené Avalon master porty jsou opět dva, stejně jako v předchozí realizaci. Počet vytvořených třiceti dvou bitových násobiček je však poloviční.

	Počet
Násobička (32bitová)	8
Avalon master port	2

Tabulka 8.11: Specifické hardwarové zdroje FIR filtru s poloviční počtem násobiček

V tabulce 8.12, v níž je uveden souhrn využitých zdrojů je vidět, že oproti předchozí paralelní realizaci je spotřebováno menší množství logických elementů a vestavěných devíti bitových násobiček. Přitom rychlost za jakou jsou data zpracována, se téměř nezměnila.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	5360/24624	22%	+39%
Kombinační funkce (LUT)	3419/24624	14%	4464/24624	18%	+31%
Vyhrazené logické registry	2314/24624	9%	3449/24624	14%	+49%
Registry	2439		3574		+47%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	18/132	14%	+350%
PLL	1/4	25%	1/4	25%	

Tabulka 8.12: Využité zdroje pro FIR filtru s poloviční počtem násobiček

Srovnání softwarového řešení a řešení s akcelerátorem je uvedeno v tabulce 8.13. Použití této struktury FIR filtru je vhodné jak pro softwarové řešení tak především pro hardwarové. Při hardwarové realizaci se efektivně sníží počet využitých zdrojů a přitom nedojde k snížení rychlosti pro zpracovávání dat.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	1093,4	
Implementace s použitím akcelerátoru	33,19	33,0x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.13: Srovnání rychlosti pro realizaci FIR filtru poloviční počtem násobiček

Při srovnání s předchozí realizací nedošlo k zásadní změně maximální frekvence hodinových signálů. Maximální frekvence se zvýšila, pouze v řádu jednotek megahertz.

F_{max} systému	133,0 MHz
F_{max} akcelerátoru	200,6 MHz

Tabulka 8.14: Maximální frekvence pro realizaci FIR filtru poloviční počtem násobiček

Latence i CPLI jsou stejné jako při realizaci v kapitole 8.2. S tím je spojeno i vlastní zrychlení hardwarového akcelerátoru, který se při srovnání s realizací v kapitole 8.2 liší pouze nepatrně.

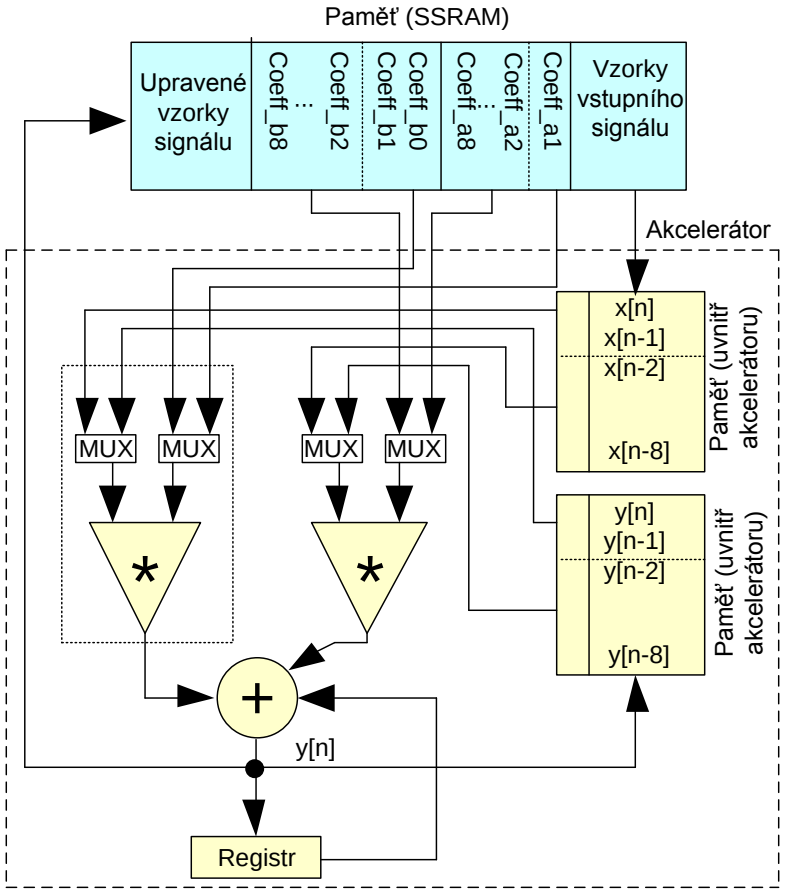
Latence	16	cyklů
CPLI	1	cyklů

Tabulka 8.15: Latence a CPLI pro realizaci FIR filtru poloviční počtem násobiček

8.4 Realizace přímé struktury IIR filtru hardwarově neoptimalizované

Realizace filtru vychází z diferenční rovnice (4), která je uvedena v kapitole 5.4. Pří použití kódu, který není uzpůsobený pro realizaci v hardwaru, C2H kompilátor vytvoří akcelerátor tak jak je zjednodušeně znázorněno na obrázku 8.4.

Takto vytvořený akcelerátor podobně jako u neoptimalizované realizace FIR filtru v kapitole 8.1 obsahuje paměť vytvořenou uvnitř akcelerátoru. Pomocí této paměti je zajišťováno zpožďování vstupních vzorků načítaných z pamětí SSRAM prostřednictvím Avalon master portu. Navíc je však vytvořena paměť pro zpožďování vzorků zpracovaného výstupního signálu. Tyto zpožďované vzorky jsou postupně násobeny koeficienty filtru, které jsou uloženy v paměti SSRAM. Výstupní hodnoty z násobiček jsou pak postupně sčítány pro všechny násobené koeficienty a vzorky uložené v paměti uvnitř akcelerátoru. Takto vypočtené výstupní hodnoty jsou uloženy do paměti, která je vytvořena v akcelerátoru a do SSRAM.



Obrázek 8.4: Realizace přímé struktury IIR filtru bez hardwarové optimalizace

Vlastní kód se skládá z hlavní a jedné vnořené smyčky. Hlavní smyčka stejně jako u realizace FIR filtru zajišťuje postupné načítání jednotlivých vzorků do paměti vytvořené v akcelerátoru i ukládání zpracovaných vzorků do paměti SSRAM. Vnořená smyčka vykonává vlastní zpožďování vstupních a výstupních vzorků a jejich násobení s koeficienty filtru. Z obrázku 8.4 a tabulky 8.16 je vidět, že v akcelerátoru jsou vytvořeny dvě třiceti dvou bitové násobičky. Přitom jedna je vytvořena pro hlavní smyčky, v které je prováděné násobení vstupního vzorku koeficientem b_0. Druhá násobička je vytvořena pro vnořenou smyčku.

	Počet
Násobička (32bitová)	2
Avalon master port	8

Tabulka 8.16: Specifické hardwarové zdroje přímé struktury IIR filtru bez optimalizace

V tabulce 8.17 je uvedeno množství spotřebovaných zdrojů při realizaci hardwarového akcelerátoru.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	5906/24624	24%	+54%
Kombinační funkce (LUT)	3419/24624	14%	4929/24624	20%	+44%
Vyhrazené logické registry	2314/24624	9%	3962/24624	16%	+71%
Registry	2439		4087		+68%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	8/132	6%	+100%
PLL	1/4	25%	1/4	25%	

Tabulka 8.17: Využité zdroje přímé struktury IIR filtru bez hardwarové optimalizace

Takto vytvořený neoptimalizovaný akcelerátor pro hardwarovou realizaci umožní 7,6x rychlejší zpracování dat oproti samostatnému soft-core procesoru.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	4736,99	
Implementace s použitím akcelerátoru	621,51	7,6x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.18: Srovnání rychlosti přímé struktury IIR filtru bez hardwarové optimalizace

F_{max} systému	125,3 MHz
F_{max} akcelerátoru	157,3 MHz

Tabulka 8.19: Maximální frekvence přímé struktury IIR filtru bez hardwarové optimalizace

V kódu pro akcelerátor se vyskytuje vnořená smyčka, ta je implementována jako samostatný stavový automat a její latence a CPLI je vyhodnocená zvlášť. V tabulce 8.20 jsou uvedeny zjištěné latence a CPLI obou smyček.

Latence hlavní smyčka	23	cyklů
Latence vnořená smyčka 1	15	cyklů
CPLI hlavní smyčka	19	cyklů
CPLI vnořená smyčka	12	cyklů

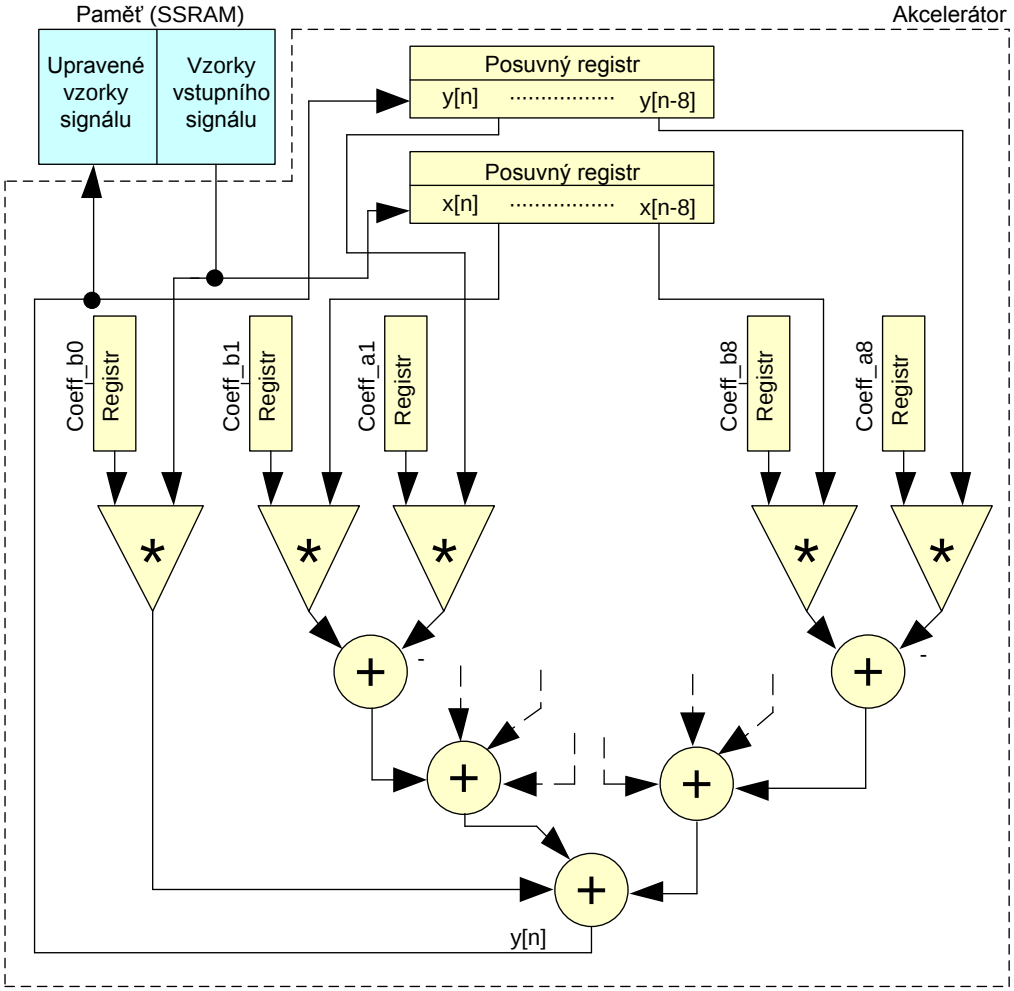
Tabulka 8.20: Latence a CPLI přímé struktury IIR filtru bez hardwarové optimalizace

8.5 Realizace přímé struktury IIR filtru optimalizované pro hardware

Optimalizací kódu přímé struktury IIR filtru pro C2H kompilátor, lze přímou strukturu filtru realizovat tak, jak je naznačeno na obrázku 8.5. Provedené optimalizace jsou stejné jako u FIR filtru v kapitole 8.2. Navíc je však vytvořen posuvný registr pro zpožďování zpracovaných vzorků výstupního signálu. Zpožděné výstupní vzorky násobené koeficienty jsou odečítány od zpožděných vstupních vzorků vynásobených koeficienty.

Pro takto vytvořený akcelerátor se vytvoří šestnáct třiceti dvou bitových násobiček a dva Avalon master porty. Přitom stejně jako u optimalizovaného FIR filtru je jeden Avalon master port pro načítání vstupních vzorků z paměti SSRAM a druhý pro ukládání zpracovaných vzorků.

V tabulce 8.21 je uveden celkový přehled využitých zdrojů. Při srovnání s neoptimalizovanou realizací byl nárůst využitých logických elementů o čtyři procenta menší a o sedm set procent vyšší u devíti bitových násobících bloků.



Obrázek 8.5: Realizace přímé struktury IIR filtru optimalizované pro hardware

	Počet
Násobička (32bitová)	16
Avalon master port	2

Tabulka 8.21: Specifické hardwarové zdroje optimalizované přímé struktury IIR filtru

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	5775/24624	23%	+50%
Kombinační funkce (LUT)	3419/24624	14%	4753/24624	19%	+39%
Vyhrazené logické registry	2314/24624	9%	3722/24624	15%	+61%
Registry	2439		3847		+58%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	36/132	27%	+800%
PLL	1/4	25%	1/4	25%	

Tabulka 8.22: Využité zdroje optimalizované přímé struktury IIR filtru

Oproti hardwarově neoptimalizovanému akcelerátoru je dosaženo 12,6x rychlejšího zpracování dat a oproti softwarové realizaci v soft-core procesoru 32,5x rychlejšího zpracování dat.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	1597,04	
Implementace s použitím akcelerátoru	49,14	32,5x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.23: Srovnání rychlosti optimalizované přímé struktury IIR filtru

Maximální frekvence samostatného akcelerátoru je o 48,5 MHz vyšší při srovnání s akcelerátorem neoptimalizovaným pro hardwarovou realizaci. Maximální frekvence hodin vstupujících do celého systému se však zvýšila pouze o 6,7 MHz.

F_{max} systému	132,0 MHz
F_{max} akcelerátoru	205,8 MHz

Tabulka 8.24: Maximální frekvence optimalizované přímé struktury IIR filtru

Latence	15	cyklů
CPLI	5	cyklů

Tabulka 8.25: Latence a CPLI optimalizované přímé struktury IIR filtru

8.6 Realizace první kanonické struktury IIR filtru hardwarově neoptimalizovaného

Realizace filtru vychází ze struktury zobrazené na obrázku 5.6. Vlastní realizace je podobná realizaci přímé struktury zjednodušeně zobrazené na obrázku 8.4. Oproti přímé struktuře IIR filtru je v akcelerátoru vytvořena pouze jedna paměť, do které jsou ukládány vypočtené vnitřní stavy. Ty jsou přičítány k vstupním a výstupním vzorkům signálu vynásobeného příslušným koeficientem.

Kód takto realizovaného akcelerátoru je opět složen z jedné hlavní smyčky a jedné vnořené. Pro vnořenou smyčku je použita jedna třiceti dvou bitová násobička a pro hlavní smyčku jsou vytvořeny dvě třiceti dvou bitové násobičky.

	Počet
Násobička (32bitová)	3
Avalon master port	7

Tabulka 8.26: Specifické hardwarové zdroje první kanonické struktury bez optimalizace

Oproti přímé struktuře IIR filtru, která není optimalizovaná pro implementaci v hardwaru je nárůst využitých logických elementů o patnáct procent menší. Celkový přehled využitých zdrojů je zobrazen v tabulce 8.27.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	5334/24624	22%	+39%
Kombinační funkce (LUT)	3419/24624	14%	4535/24624	18%	+33%
Vyhrazené logické registry	2314/24624	9%	3428/24624	14%	+48%
Registry	2439		3553		+46%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	10/132	8%	+150%
PLL	1/4	25%	1/4	25%	

Tabulka 8.27: Využité zdroje první kanonické struktury IIR filtru bez optimalizace

Srovnání rychlosti zpracování dat filtrem IIR první kaskádní struktury realizovaného softwarově a implementace filtru jako hardwarového akcelerátoru je uvedeno v tabulce 8.28. Filtr realizovaný jako hardwarový akcelerátor umožňuje 8,8x rychlejší zpracování dat oproti softwarové realizaci. Oproti realizaci IIR filtru přímé struktury bez hardwarové optimalizace jsou data zpracována 1,8x rychleji.

	Doba zpracování* [μs]	Zrychlení
Implementace pro CPU Nios II	2986,26	
Implementace s použitím akcelerátoru	338,03	8,8x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.28: Srovnání rychlosti první kanonické struktury IIR filtru bez optimalizace

F_{max} systému	123,0 MHz
F_{max} akcelerátoru	196,7 MHz

Tabulka 8.29: Maximální frekvence první kanonické struktury IIR filtru bez optimalizace

V kódu se vyskytuje vnořená smyčka proto je opět latence a CPLI udáno pro každou smyčku zvlášť.

Latence hlavní smyčka	19	cyklů
Latence vnořená smyčka 1	16	cyklů
CPLI hlavní smyčka	7	cyklů
CPLI vnořená smyčka	7	cyklů

Tabulka 8.30: Latence a CPLI první kanonické struktury IIR filtru bez optimalizace

8.7 Realizace první kanonické struktury IIR filtru optimalizované pro hardware

Vlastní realizace první kanonické struktury optimalizované pro hardwarovou implementaci je podobná struktuře zobrazené na obrázku 5.6. Zpožďovací členy jsou stejně jako koeficienty uložené přímo v hradlovém poli ve vytvořených registrech. Počet vytvořených třiceti dvou bitových násobiček odpovídá dvounásobku řádu filtru. Pro každý koeficient násobený vstupním nebo výstupním signálem je vytvořena jedna násobička. Pro přístup do SSRAM paměti je vytvořen jeden master port společný jak pro načítání, tak pro ukládání vzorků do paměti.

	Počet
Násobička (32bitová)	16
Avalon master port	1

Tabulka 8.31: Specifické hardwarové zdroje optimalizované první kanonické struktury

Při srovnání s přímou strukturou optimalizovanou pro hardwarovou realizaci je nárůst spotřebovaných logických elementů o 29% nižší a oproti první kanonické struktuře bez optimalizace o 18% nižší. Celkový přehled je uveden v tabulce 8.32.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	4641/24624	19%	+21%
Kombinační funkce (LUT)	3419/24624	14%	4133/24624	17%	+21%
Vyhrazené logické registry	2314/24624	9%	2756/24624	11%	+19%
Registry	2439		2881		+18%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	28/132	21%	+600%
PLL	1/4	25%	1/4	25%	

Tabulka 8.32: Využité zdroje optimalizované první kanonické struktury IIR filtru

Oproti hardwarově neoptimalizovanému akcelerátoru je dosaženo 5,9x rychlejší zpracování dat.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	1728,35	
Implementace s použitím akcelerátoru	57,77	29,9x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.33: Srovnání rychlosti optimalizované první kanonické struktury IIR filtru

Optimalizací pro hardwarovou realizaci se maximální frekvence pro vlastní akcelerátor téměř nezměnila.

Fmax systému	146,2 MHz
Fmax akcelerátoru	192,9 MHz

Tabulka 8.34: Maximální frekvence optimalizované první kanonické struktury IIR filtru

Latence	12	cyklů
CPLI	9	cyklů

Tabulka 8.35: Latence a CPLI optimalizované první kanonické struktury IIR filtru

8.8 Realizace kaskádní struktury IIR filtru hardwarově neoptimalizovaného

Realizace vychází ze struktury zobrazené na obrázku 5.7. Pro jednotlivé přenosy sekcí druhého řádu je použita první kanonická struktura. Vlastní kód opět obsahuje hlavní a jednu vnořenou smyčku. Vnořená smyčka realizuje během jedné iterace výpočet celé jedné sekce druhého řádu. Při realizaci v hradlovém poli, odpovídající vnořené smyčce, jsou vytvořeny tři třiceti dvou bitové násobičky. Jedna třiceti dvou bitová násobička je vytvořena pro hlavní smyčku.

	Počet
Násobička (32bitová)	4
Avalon master port	5

Tabulka 8.36: Specifické hardwarové zdroje kaskádní struktury bez optimalizace

Ze všech realizací bylo spotřebováno největší množství logických elementů, které z celkového množství logických elementů použitého hradlového pole činí 25%, z toho je 9% spotřebována na samotný akcelerátor. Celkový přehled je uveden v tabulce 8.37.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	6060/24624	25%	+58%
Kombinační funkce (LUT)	3419/24624	14%	5006/24624	20%	+46%
Vyhrazené logické registry	2314/24624	9%	3956/24624	16%	+71%
Registry	2439		4081		+67%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	16/132	12%	+300%
PLL	1/4	25%	1/4	25%	

Tabulka 8.37: Využité zdroje kaskádní struktury IIR filtru bez optimalizace

Rychlost za jakou jsou testovací data zpracována je při použití akcelerátoru oproti samotnému soft-core procesoru 14,1x větší.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	4470,87	
Implementace s použitím akcelerátoru	318,10	14,1x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.38: Srovnání rychlosti kaskádní struktury IIR filtru bez optimalizace

F_{max} systému	120,0 MHz
F_{max} akcelerátoru	166,6 MHz

Tabulka 8.39: Maximální frekvence kaskádní struktury IIR filtru bez optimalizace

Opět jsou latence i CPLI pro každou smyčku vyhodnoceny zvlášť.

Latence hlavní smyčka	16	cyklů
Latence vnořená smyčka 1	20	cyklů
CPLI hlavní smyčka	2	cyklů
CPLI vnořená smyčka	12	cyklů

Tabulka 8.40: Latence a CPLI kaskádní struktury IIR filtru bez optimalizace

8.9 Realizace kaskádní struktury IIR filtru optimalizovaného pro hardware

Kaskádní struktura optimalizovaná pro hardwarovou realizaci vychází ze struktury zobrazené na obrázku 5.7. Jednotlivé sekce druhého řádu jsou realizovány jako první kanonická struktura. V každé sekci jsou pro ukládání vnitřních stavů a koeficientů v hradlovém poli vytvořeny registry. Počet vytvořených třiceti dvou bitových násobiček je oproti předchozím realizacím vyšší, což je zapříčiněno násobením navíc, vzorků vystupujících z jednotlivých sekcí váhovými koeficienty. Vytvořené Avalon master porty jsou dva a to pro načítání a zápis da paměti SSRAM.

	Počet
Násobička (32bitová)	21
Avalon master port	2

Tabulka 8.41: Specifické hardwarové zdroje optimalizované kaskádní struktury

Celkový přehled využitých zdrojů hradlového pole je uveden v tabulce 8.42.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	5329/24624	22%	+39%
Kombinační funkce (LUT)	3419/24624	14%	4602/24624	19%	+35%
Vyhrazené logické registry	2314/24624	9%	3328/24624	14%	+44%
Registry	2439		3453		+42%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	67/132	51%	+1575%
PLL	1/4	25%	1/4	25%	

Tabulka 8.42: Využité zdroje optimalizované kaskádní struktury IIR filtru

Realizovaný akcelerátor umožňuje oproti softwarovému řešení 56,0x rychlejší zpracování dat.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	2270,56	
Implementace s použitím akcelerátoru	40,57	56,0x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.43: Srovnání rychlosti optimalizované kaskádní struktury IIR filtru

F_{max} systému	123,3 MHz
F_{max} akcelerátoru	173,9 MHz

Tabulka 8.44: Maximální frekvence optimalizované kaskádní struktury IIR filtru

Takto realizovaný akcelerátor dosahuje ze všech akcelerátorů optimalizovaných pro hardwarovou implementaci největší zpoždění vstupního signálu. Jeho latence je 33 cyklů.

Latence	33	cyklů
CPLI	4	cyklů

Tabulka 8.45: Latence a CPLI optimalizované kaskádní struktury IIR filtru

8.10 Realizace paralelní struktury IIR filtru

Následující realizace vychází se struktury zobrazené na obrázku 5.8. Stejně jako u kaskádní struktury jsou jednotlivé sekce druhého řádu realizované jako první kanonická struktura. Využitých třiceti dvou bitových násobiček je méně jako v případě kaskádní struktury. To je způsobeno tím, že váhové koeficienty byli započteny přímo do

jednotlivých přenosů druhého řádu. Tím odpadla nutnost zvlášť násobit vzorky vstupující do jednotlivých sekcí. Jediná násobička, která vznikne navíc, je umístěna za sčítačkou, která provádí součet vzorků z jednotlivých výstupní sekcí. Tato násobička násobí výstupní signál takovou konstantou, o kterou byl snížen vstupní signál vlivem váhového koeficientu, který je započítán ve všech dílčích přenosech. Přehled vytvořených třiceti dvou bitových násobiček a Avalon master portů je uveden v tabulce 8.46.

	Počet
Násobička (32bitová)	17
Avalon master port	2

Tabulka 8.46: Specifické hardwarové zdroje paralelní struktury

Celkový přehled spotřebovaných zdrojů je uveden v tabulce 8.47. Oproti kaskádní struktuře je nárůst spotřebovaných logických elementů o 10% nižší a u devíti bitových bloků nižší o 975%.

	Využité zdroje pouze pro CPU Nios II		Využité zdroje pouze pro CPU Nios II a akcelerátor		Nárůst
Logické elementy	3844/24624	16%	4940/24624	20%	+29%
Kombinační funkce (LUT)	3419/24624	14%	4263/24624	17%	+25%
Vyhrazené logické registry	2314/24624	9%	3035/24624	12%	+31%
Registry	2439		3160		+30%
Piny	68/216	31%	68/216	31%	
Paměťové bity	65088/608256	11%	65088/608256	11%	
Vestavěné 9bitové násobičky	4/132	3%	28/132	21%	+600%
PLL	1/4	25%	1/4	25%	

Tabulka 8.47: Využité zdroje paralelní struktury IIR filtru

Akcelerátor umožňuje oproti softwarovému řešení 46,3x rychlejší zpracování dat. Jednotlivé časy pro zpracování testovacích dat jsou uvedeny v tabulce 8.48.

	Doba zpracování* [µs]	Zrychlení
Implementace pro CPU Nios II	2068,29	
Implementace s použitím akcelerátoru	44,63	46,3x
* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu		

Tabulka 8.48: Srovnání rychlosti paralelní struktury IIR filtru

F_{max} systému	128,8 MHz
F_{max} akcelerátoru	196,4 MHz

Tabulka 8.49: Maximální frekvence paralelní struktury IIR filtru

Při srovnání s kaskádní strukturou je latence téměř poloviční při stejném CPLI. Zjištěné hodnoty latence a CPLI pro paralelní strukturu jsou uvedeny v tabulce 8.50.

Latence	17	cyklů
CPLI	4	cyklů

Tabulka 8.50: Latence a CPLI paralelní struktury IIR filtru

8.11 Testování funkčnosti implementovaných filtrů

Pro testování funkčnosti implementovaných struktur filtru byl vytvořený v programu Matlab, skript. Tento skript umožňuje generování testovacího signálu, který může být harmonický signál o různých frekvencích, bílý šum případně sinus sweeps signál. Jednotlivé vzorky vygenerovaného testovacího signálu jsou uloženy do souboru odděleny čárkou. Pro načtení dat z tohoto souboru do paměti vývojového kitu je použit Host-Based File Systém, který umožňuje číst a zapisovat data do zvoleného souboru z hostitelského počítače. Data jsou přenášena přes kabel, který je používán pro konfiguraci hradlového pole a nahrávání programu. Pro přístup k datům se využívají standartní I/O funkce `fopen`, `fprintf`, `fclose` atd. Po zpracování těchto vzorků implementovaným algoritmem jsou upravené vzorky pomocí Host-Based File systému uloženy do souboru v hostitelském počítači. Jednotlivé vzorky jsou opět odděleny čárkou.

Při opětovném spuštění skriptu v programu Matlab jsou načteny vzorky upraveného signálu ze souboru. Upravený signál je pak zobrazen společně s původním a současně je zobrazeno jeho amplitudové spektrum. Pro testování implementovaných algoritmů byl především využíván harmonický signál. Konkrétní implementací filtru bylo pak zpracováno několik harmonických signálů na určitých kontrolních frekvencích. Takto zpracované signály pak byly porovnány s předpokládaným výstupní signálem z filtru.

8.12 Souhrn dosažených výsledků

Uváděné CPLI a Latence implementovaných struktur nezahrnují časy strávené vykonáváním vnořených smyček, volání subfunkcí a přístupu do paměti. Z tohoto důvodu je možné že struktury, které mají stejnou latenci i CPLI mohou vykazovat nepatrně rozdílné časy při zpracování dat

Z uvedených realizací FIR filtrů nejlepších vlastností dosahuje struktura s polovičním počtem násobiček a to jak pro softwarovou tak hardwarovou realizaci. U této struktury bylo dosaženo nejrychlejšího zpracování dat. Současně bylo oproti základní struktuře FIR filtru optimalizované pro hardwarovou implementaci spotřebováno menší množství logických elementů a o polovinu méně devíti bitových násobících bloků pro vytvořený akcelerátor. Struktura umožňuje teoreticky zpracovat jeden vzorek během jednoho hodinového cyklu s latencí šestnácti hodinových cyklů. U

filtru FIR je složitost implementace u všech struktur téměř stejná, koeficienty filtru vždy vycházejí menší jak jedna a je tedy snadné je převést do fixed-point tvaru.

Pro realizaci filtru IIR v hradlovém poli jsou nejvhodnější struktury kaskádní a paralelní z důvodu jednoduššího převedení koeficientů do tvaru fixed-point. Při zvolení nevhodné struktury může dojít vlivem velikostně příliš rozdílných koeficientů k značnému znesnadnění celé implementace filtru v hradlovém poli. Výhodou paralelní struktury je možnost dosažení malého CPLI se současně relativně nízkou latencí. Při srovnání s kaskádní strukturou je latence paralelní struktury téměř dvojnásobně menší při stejném CPLI.

U implementovaných struktur bylo zjištěno, že lze dosáhnout zrychlení 8,3x až 14,1x většího vůči samotnému soft-core procesoru Nios II, pouze jednoduchými úpravami kódu určeného pro softwarovou realizaci. Jednalo se o úpravy typu použití propojovací direktivy paměti a použití klíčového slova restrict u ukazatelů. Při optimalizaci kódu pro realizaci v hardwaru bylo dosaženo zrychlení 29,9x až 56,0x většího vůči samotnému procesoru Nios II.

Celkový přehled všech implementovaných struktur filtru jako akcelerátoru a jejich zrychlení vůči samotnému soft-core procesoru, spolu s využitými zdroji hradlového pole a maximální možnou frekvencí, latencí a CPLI jsou uvedeny v tabulce v příloze 2. Z této tabulky je názorně vidět jaký má vliv uzpůsobení kódu pro hardwarovou realizaci.

9 IMPLEMENTACE HRANOVÉHO OPERÁTORU

Vytvořený příklad pro zpracování obrazu pomocí hranových operátorů, umožňuje zobrazení výsledného obrazu na LCD displeji vývojového kitu. Vlastní obraz, který má být zpracován je uložen do paměti SDRAM ve formátu RGB. Pro každý pixel je tedy uložena intenzita základních tří barev červené, zelené a modré. Skládáním těchto tří základních barev pak vznikají ostatní odstíny. Pro uložení intenzity každé ze tří základních barev je použito osmi bitů. Intenzita tedy může nabývat hodnoty od 0 do 255. Pro uložení jednoho pixelu je tedy zapotřebí dvacet čtyři bitů.

Takto uložený obraz je nejprve převeden na šedotónový pomocí vztahu (11), v kterém Y je odstín šedi a R, G, B jsou intenzity tři základních barev. Pro rychlejší zpracování v procesoru Nios II je vztah (11) převeden do aritmetiky s pevnou řádovou čárkou.

$$Y = 0{,}2989 \cdot R + 0{,}5870 \cdot G + 0{,}1140 \cdot B \qquad (11)$$

Na takto získaný šedotónový obraz jsou aplikovány hranové operátory. Ve vytvořeném příkladu je na výběr horizontální a vertikální Sobelův operátor a Laplacián. Všechny použité lokální operátory mají rozměry matice 3x3.

Vlastní 2D konvoluce, masky a obrazu je realizovaná jako hardwarový akcelerátor. V tomto případě však nedošlo k požadovanému zrychlení. Problém je v tom, že akcelerátor při násobení koeficientů masky načítal jednotlivé pixely obrazu uloženého v datové paměti procesoru (SDRAM). Tyto časté přístupy do paměti jsou slabinou, která znemožňuje vlastní zrychlení oproti samotnému procesoru Nios II. Řešením tohoto problému je vytvoření pamětí v hradlovém poli, do kterých by byly během výpočtu načítány jednotlivé řádky obrazu z paměti SDRAM. Teprve až pixely uložené v paměti vytvořené v hradlovém by byly násobeny koeficienty masky. V příloze 3 jsou zobrazeny obrázky s vývojovým kitem a vytvořeným příkladem.

10 ZÁVĚR

V rámci práce bylo implementováno několik struktur číslicových filtrů s využitím techniky C2H, která umožňuje vytváření hardwarových akcelerátorů připojených k soft-core procesoru Nios II, z funkcí napsaných v jazyce ANSI C.

U implementovaných struktur bylo zjištěno, že lze dosáhnout pouze jednoduchými úpravami kódu určeného pro softwarovou realizaci, zrychlení 8,3x až 14,1x většího vůči samotnému soft-core procesoru Nios II. Jedná se o úpravy typu použití propojovací direktivy paměti a použití klíčového slova restrict u ukazatelů. Při optimalizaci kódu pro realizaci v hardwaru bylo dosaženo zrychlení 29,9x až 56,0x většího vůči samotnému procesoru Nios II. Nárůst využitých logických elementů se u implementovaných algoritmů pohyboval od 21% až po 58%. Nejpodstatnější nárůst je u využitých devíti bitových násobících bloků, který se u jednotlivých implementací pohybuje ve stovkách procent.

Také byl realizován příklad pro zpracování obrazu pomocí hranových operátorů, který umožňuje zobrazení zpracovaného obrazu na LCD displeji. V tomto případě však nedošlo k požadovanému zrychlení. Důvod proč k zrychlení nedošlo, je ten, že na akcelerátor byl kladen požadavek na současné zpracování více dat uložených v datové paměti procesoru. Omezení zrychlení bylo tedy způsobeno dostupností vlastních dat. Řešením toho to problému by bylo použití pamětí vytvořených v hradlovém poli pro ukládání části dat při zpracování. Z časových důvodů však k dané úpravě již nedošlo.

Využitím techniky C2H u některých typů algoritmů, jako jsou například 1D filtry realizované v této práci, lze dosáhnout podstatného zrychlení oproti samotnému procesoru Nios II. Přitom u složitějších projektů, u kterých je vyžadovaná i činnost procesoru, odpadá nutnost při použití této techniky vytváření samostatného hardwarového modulu v HDL jazyce a řešení jeho připojení k procesoru.

11 POUŽITÁ LITERATURA

[1] ALTERA. *Nios II C2H Compiler*: *User Guide* [online]. 9.1. San Jose, November 2009 [cit. 2012-05-11]. Dostupné z: http://www.altera.com/literature/ug/ug_nios2_c2h_compiler.pdf

[2] ALTERA. *SOPC Builder*: *User Guide* [online]. San Jose, December 2010 [cit. 2012-05-11]. Dostupné z: http://www.altera.com/literature/ug/ug_sopc_builder.pdf

[3] *Automated Generation of Hardware Accelerators With Direct Memory Access From ANSI/ISO Standard C Functions* [online]. 1.0. May 2006[cit. 2012-05-11]. Dostupné z: http://www.altera.com/literature/wp/wp-aghrdwr.pdf

[4] ALTERA. *Nios II Embedded Evaluation Kit, Cyclone III Edition*: *User Guide* [online]. San Jose, July 2010 [cit. 2012-05-11]. Dostupné z: http://www.altera.com/literature/ug/niosii_eval_user_guide.pdf

[5] Cyclone III FPGA: Architecture. ALTERA. *Altera*: *FPGAs* [online]. [cit. 2012-05-10]. Dostupné z: http://www.altera.com/devices/fpga/cyclone3/overview/architecture/cy3-architecture.html

[6] ALTERA. *Cyclone III Device Handbook*: *Volume 1* [online]. 11.1. San Jose, December 2011 [cit. 2012-05-11]. Dostupné z: http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf

[7] JAN, Jiří. *Číslicová filtrace, analýza a restaurace signálů*. Brno: VUTIUM, 2002. ISBN 80-214-1558-4.

[8] SOVKA, Pavel, Roman ČMEJLA a Ladislav ŠMEJKAL. Číslicové filtry II. *Automatizace* [online]. 2005, roč. 48, č. 9 [cit. 2012-05-11]. Dostupné z: http://www.automatizace.cz/article.php?a=838

[9] SMÉKAL, Zdeněk a SYSEL. *Číslicové filtry* [online]. 8.12.2004. [cit. 2012-05-11].

[10] VÍCH a Zdeněk SMÉKAL. *Číslicové Filtry*. Praha: Academia, 2000. ISBN 80-200-0761-X.

[11] DAVÍDEK, Vratislav, Miloš LAIPERT a Miroslav VLČEK. *Analogové a číslicové filtry*. 2. vyd. Praha: ČVUT, 2004. ISBN 80-01-03026-1.

[12] SOVKA, Pavel, Roman ČMEJLA a Ladislav ŠMEJKAL. Číslicové filtry I. *Automatizace* [online]. 2005, roč. 48, 7-8 [cit. 2012-05-11]. Dostupné z: http://www.automatizace.cz/article.php?a=792

[13] SOVKA, Pavel, Roman ČMEJLA a Ladislav ŠMEJKAL. Číslicové filtry III. *Automatizace* [online]. 2005, roč. 48, č. 10 [cit. 2012-05-11]. Dostupné z: http://www.automatizace.cz/article.php?a=893

[14] TIŠNOVSKÝ, Pavel. *Fixed point arithmetic* [online]. 2006[cit. 2012-05-11]. Dostupné z: http://www.root.cz/serialy/fixed-point-arithmetic

[15] FRANCIS, Michael. Infinite Impulse Response Filter Structures in Xilinx FPGAs. In: [online]. 1.2, 10.8.2009 [cit. 2012-05-11]. Dostupné z: http://www.xilinx.com/support/documentation/white_papers/wp330.pdf

[16] ALTERA. *Nios II Processor Reference*: *Handbook* [online]. 11.0. San Jose, May 2011 [cit. 2012-05-11]. Dostupné z: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

Seznam zkratek

C2H	-C to Hardvare Acceletarion kompilátor
FPGA	-programovatelné hradlové pole
SDRAM	-synchronní dynamická paměť s náhodným přístupem
SSRAM	-synchronní statická paměť s náhodným přístupem
HDL	-jazyk popisující hardware
PLL	-Phase-locked loop (fázový závěs)
LAB	-logický blok
LE	-logický element
LUT	-vyhledávací look-up tabulku
FIR	-filtr s konečnou impulsní odezvou
IIR	-filtr s nekonečnou impulsní odezvou
I/O	-Input/Output
LCD	-Liquid crystal displej
CPLI	-Cycles Per Loop Iteration

Seznam příloh

Příloha 1: Seznamy použitých hodin a sestavené procesorové systémy

Příloha 2: Přehledová tabulka

Příloha 3: Vytvořený příklad pro vývojový kit

Příloha 4: DVD

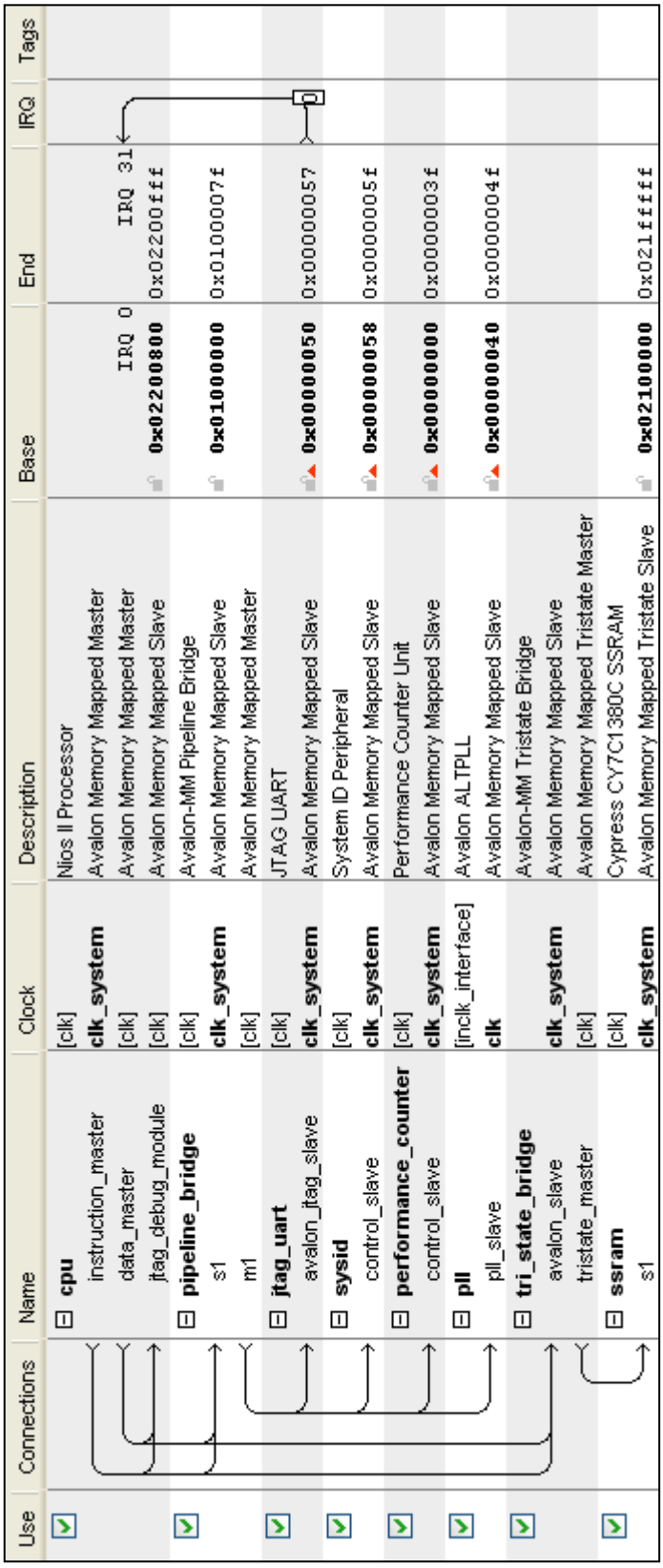
Příloha 1: Seznamy použitých hodin a sestavené procesorové systémy

Seznam hodin v procesorovém systému pro lineární číslicové filtry

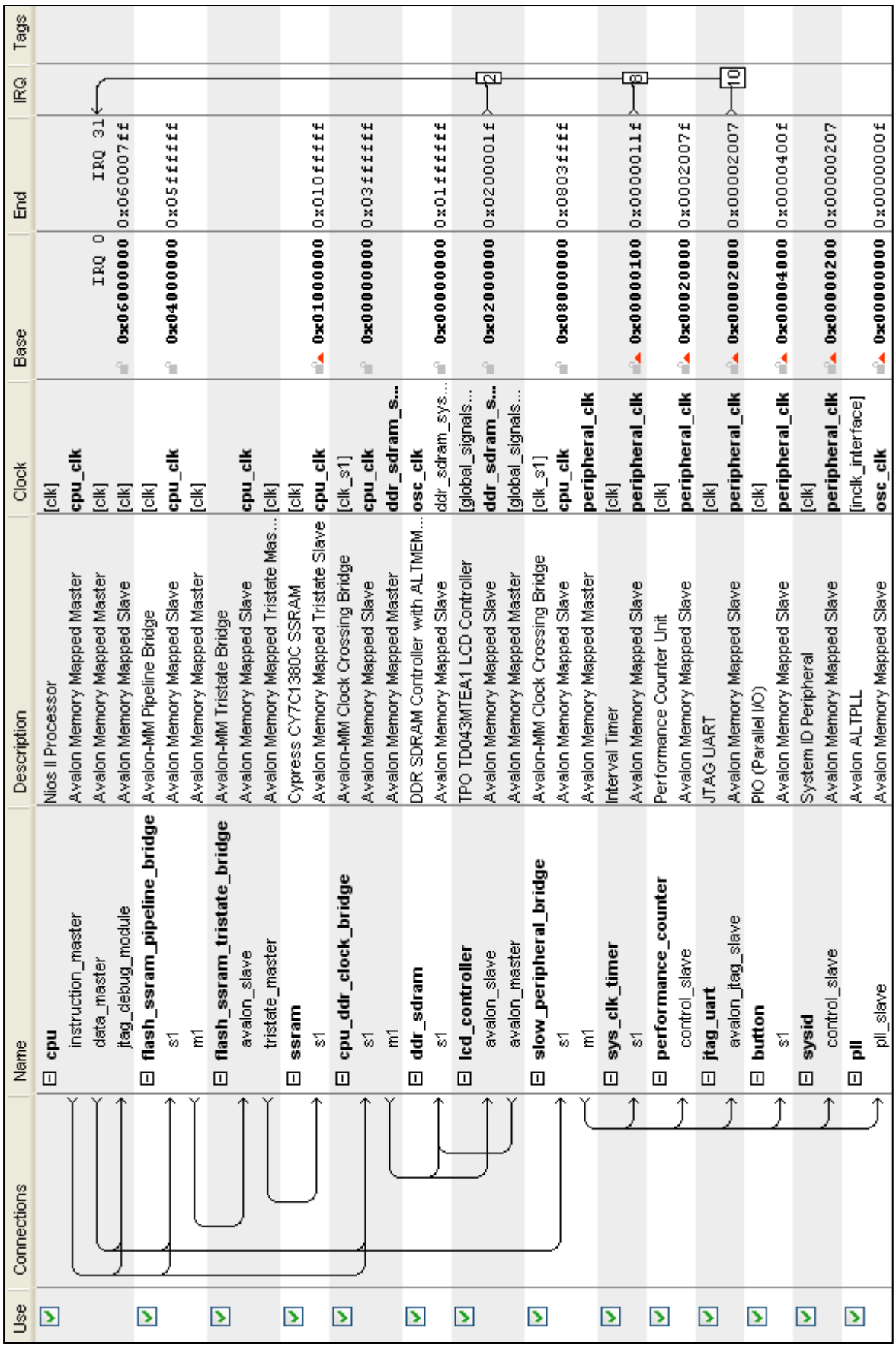
Name	Source	MHz
clk	**External**	**50,0**
clk_system	**pll.c0**	100,0
clk_ssram	**pll.c1**	100,0

Seznam hodin v procesorovém systému pro hranové operátory

Name	Source	MHz
osc_clk	**External**	**50,0**
ddr_sdram_sysclk	**ddr_sdram.sysclk**	66,5
ddr_sdram_auxfull	**ddr_sdram.auxfull**	133,0
ddr_sdram_auxhalf	**ddr_sdram.auxhalf**	66,5
cpu_clk	**pll.c0**	100,0
ssram_clk	**pll.c1**	100,0
peripheral_clk	**pll.c2**	80,0
remote_update_clk	**pll.c3**	40,0


Use
Connections
Name
Clock
Description
Base
End
IRQ
Tags
cpu
[clk]
Nios II Processor
IRQ 0
IRQ 31
instruction_master
clk_system
Avalon Memory Mapped Master
data_master
[clk]
Avalon Memory Mapped Master
0x02200800
0x02200fff
jtag_debug_module
[clk]
Avalon Memory Mapped Slave
pipeline_bridge
[clk]
Avalon-MM Pipeline Bridge
0x01000000
0x0100007f
s1
clk_system
Avalon Memory Mapped Slave
m1
[clk]
Avalon Memory Mapped Master
jtag_uart
[clk]
JTAG UART
0x00000050
0x00000057
avalon_jtag_slave
clk_system
Avalon Memory Mapped Slave
sysid
[clk]
System ID Peripheral
0x00000058
0x0000005f
control_slave
clk_system
Avalon Memory Mapped Slave
performance_counter
[clk]
Performance Counter Unit
0x00000000
0x0000003f
control_slave
clk_system
Avalon Memory Mapped Slave
pll
[inclk_interface]
Avalon ALTPLL
0x00000040
0x00000043f
pll_slave
clk
Avalon Memory Mapped Slave
tri_state_bridge
Avalon-MM Tristate Bridge
avalon_slave
clk_system
Avalon Memory Mapped Slave
tristate_master
[clk]
Avalon Memory Mapped Tristate Master
ssram
[clk]
Cypress CY7C1380C SSRAM
0x02100000
0x021fffff
s1
clk_system
Avalon Memory Mapped Tristate Slave

Procesorový systém pro hranové operátory s možností zobrazení na LCD



Příloha 2: Přehledová tabulka

Realizace	Doba zpracování* [µs]	Zrychlení [-] (oproti samotnému Nios II)	Logické elementy [-]	9bitové násobičky [-]	F_{max} [MHz] akcelerátoru	Latence [cyklů]	CPLI [cyklů]
FIR filtru s jednou násobičkou	659,82	8,3x	5347/24624	6/132	199,6		
Paralelní realizace FIR filtru	35,66	42,2x	5608/24624	32/132	192,1	16	1
Paralelní realizace FIR filtru s polovičním počtem násobiček	33,19	33,0x	5360/24624	18/132	200,6	16	1
Realizace přímé struktury IIR filtru hardwarově neoptimalizovaného	621,51	7,6x	5906/24624	8/132	157,3		
Realizace přímé struktury IIR filtru optimalizované pro hardware	49,14	32,5x	5775/24624	36/132	205,8	15	5
Realizace první kanonické struktury IIR filtru hardwarově neoptimalizovaného	338,03	8,8x	5334/24624	10/132	196,7		
Realizace první kanonické struktury IIR filtru optimalizované pro hardware	57,77	29,9x	4641/24626	28/132	192,9	12	9
Realizace kaskádní struktury IIR filtru hardwarově neoptimalizovaného	318,10	14,1x	6060/24626	19/132	166,6		
Realizace kaskádní struktury IIR filtru optimalizovaného pro hardware	40,57	56,0x	5329/24626	67/132	173,9	33	4
Realizace paralelní struktury IIR filtru	44,63	46,3x	4940/24626	28/132	196,39	17	4

* Doba zpracovaní je uvedena pro 440 vzorků vstupního signálu

Příloha 3: Vytvořený příklad pro vývojový kit

Zobrazený originální obrázek

Obrázek převedený na šedotónový

Obrázek hranovaný Sobelovým operátorem